The TIME-LIFE Step-by-Step Guide to the IBM PCjr

Random House New York

Created and designed by
TIME-LIFE BOOKS INC.

EDITOR: George Constable
Executive Editor: George Daniels
Director of Design: Louis Klein
Director of Administration: David L. Harrison
Director of Research: Phyllis K. Wise
Director of Photography: John Conrad Weiser

PRESIDENT: Reginald K. Brack Jr.
Senior Vice President: William Henry
Vice Presidents: George Artandi, Stephen L. Bair, Robert A. Ellis, Juanita T. James, Christopher T. Linen, James L. Mercer, Joanne A. Pello, Paul R. Stewart

THE TIME-LIFE STEP-BY-STEP GUIDE TO THE IBM PCjr

Editor: Peter Pocock
Book Manager: Loretta Y. Britten
Designers: Ellen Robling, Edward Frank
Associate Editors: Sarah Brash, Roberta Conlan, Jan Leslie Cook (text)
Project Administrator: Caroline A. Boubin
Picture Editors: Barbara Moir, Marta A. Sanchez
Staff Writer: M. Linda Lee
Assistant Designer: Robert K. Herndon
Copy Coordinator: Margery duMond
Picture Coordinator: Nancy L. Scott
Special Contributors: Marilynne Rudick, Kirk Y. Saunders, Brooke Stoddard, David H. Wyatt (text)

EDITORIAL OPERATIONS
Copy Room: Diane Ullius
Production: Anne B. Landry (director), Celia Beattie
Quality Control: James J. Cox (director), Sally Collins
Library: Louise D. Forstall

© 1984 Time-Life Books Inc. All rights reserved.
No part of this book may be reproduced in any form or by any electronic or mechanical means, including information storage and retrieval devices or systems, without prior written permission from the publisher, except that brief passages may be quoted for reviews.

TIME-LIFE is a trademark of Time Incorporated U.S.A.

Published in the United States by Random House, Inc., New York, and simultaneously in Canada by Random House of Canada Limited, Toronto.

Library of Congress Cataloguing in Publication Data
Main entry under title:

The Time-Life step-by-step guide to the IBM PCjr.

Bibliography: p.
Includes index.
1. IBM PCjr (Computer). I. Time-Life Books.
QA76.8.I2593T56 1984 001.64 83-24577
ISBN 0-394-72519-0

Manufactured in the U.S.A.
98765432
First Edition

THE CONSULTANT

Isabel Lida Nirenberg has worked on a wide range of computer applications, from the analysis of data collected by the *Pioneer* space probes, to the matching of children and families for adoption agencies. She manages the DEC-20 computer system at the State University of New York at Albany, and assists faculty and students there with microcomputer applications.

PHOTOGRAPHS by Larry Sherer

A Versatile Computer System

The IBM PCjr can be the core of many different computer systems, each designed for a completely different application. An artist might use the PCjr's color and graphics features to create original images or to manipulate existing ones by removing lines, altering colors and rotating shapes. In a small office, a PCjr could be used to calculate accounts and keep financial records or to do word processing and maintain correspondence files. At home, you could use it to keep budgets, to maintain household records, to communicate with other computer users or to play games.

The reason the computer can do all these jobs is that it is designed to control a multitude of accessories used for specialized applications. Most of these accessories, called peripheral components, are made by IBM, but many more will be produced by other manufacturers for use with the IBM PCjr.

If you plan to use your PCjr for relatively modest operations, you can buy the basic entry-level model — a package containing a system unit, a cordless keyboard and an electrical transformer — and add peripherals as you need them. More complex operations, such as word processing, require the disk drive and 128K memory of the expanded PCjr system. The PCjr is designed to grow with your needs, simply through the addition of peripherals.

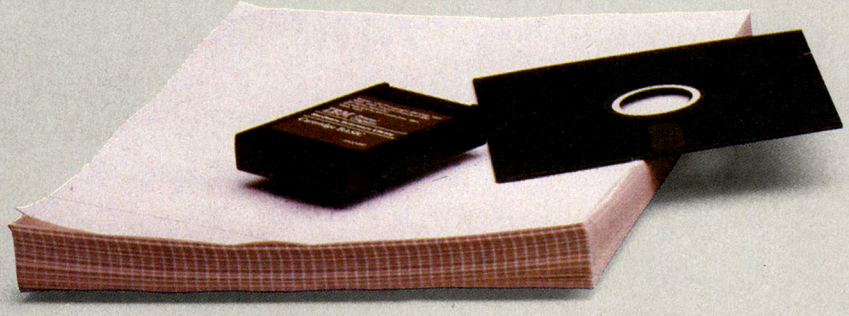
The expanded system shown here includes the components needed for the most common uses of the PCjr — word processing, budgeting, display of color graphics, games, information storage and retrieval, and the printing out of results. Your system may include some or all of these components, and others chosen to meet your particular needs. This manual will show you the basic principles of operating the IBM PCjr system and will explore some of the ways the computer can be adapted for specialized applications.

The PCjr's Parts: An Anatomy Lesson

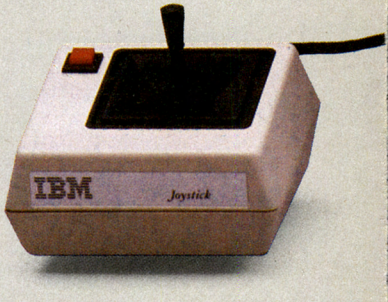
A printer produces a paper record of the computer's work. The standard IBM Compact Printer shown here produces rough but clear type on heat-sensitive paper; more specialized printers produce fine-quality print or color graphics.



Sheets of 8½-by-11-inch fanfold thermal paper are connected at the top and bottom by rows of perforations. A spring-clip release lever on the printer locks the paper in place for continuous feed. Single sheets and roll paper are also available.

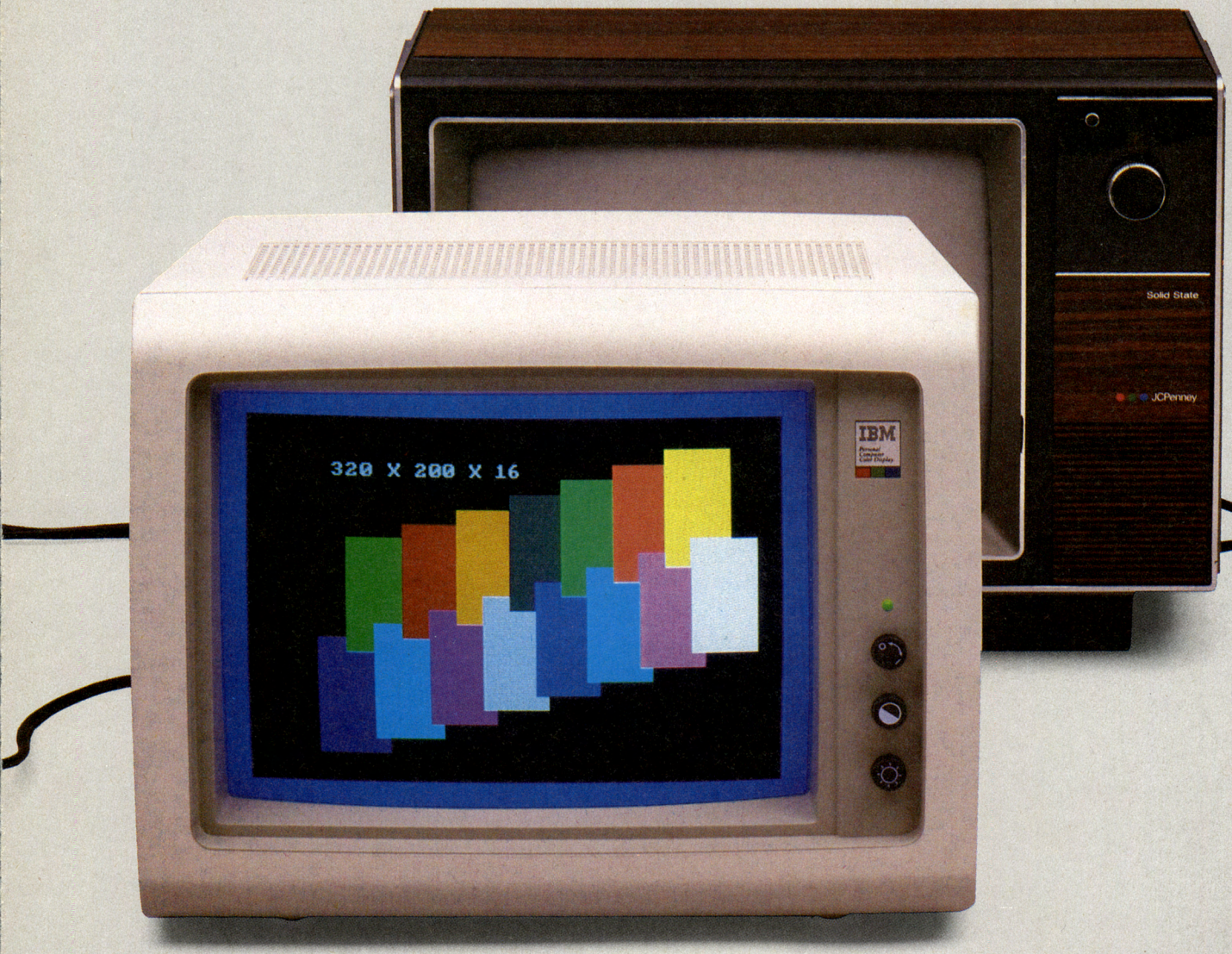


Cartridges and floppy disks contain computer programs or information. Floppy disks are also used to store information.



A joystick controls the computer for games. Usually joysticks are connected in pairs, so that more than one person can play a game at one time.

Using the household color television set to display work helps keep the cost of a system low. A color TV is usually adequate for games and educational programs.



A color video monitor is best suited for uses where color and flexibility are important, as in the preparation of charts, maps and games. Text lines displayed on the screen may be either 40 or 80 characters long.

An audio box, or external speaker, provides sound transmission when the PCjr is used with a monitor that has no speaker; when attached to a TV, the PCjr uses the set's sound system.

The system unit is the heart of the computer system, housing the electronic circuits that process and store information. The workings of all the other parts of the system are controlled by this unit.

An electrical transformer converts 110-volt alternating current into the low-voltage direct current used by the PCjr. The system draws up to 33 watts of power.



The keyboard is the device most commonly used to transfer information into the computer. Instructions typed on the keyboard can direct the computer to perform an enormous range of tasks. The PCjr's cordless keyboard communicates by infrared signals to the system unit; an optional cord is available.

Contents

The Hardware	8
Getting Started	20
The Operating System	30
Education and Entertainment	44
Programs for Productivity	58
Computer Communications	66
Programming in BASIC	74
Appendix	96
Index with Glossary	101


Esc
1
!
2
@
3
#
F1
F2
F3
Tab
Q
W
Pause
A
S
Ctrl
Shift
Z

The Hardware: Set-up to Start-up

When you unwrap your PCjr, it will be nearly ready to use. All you have to do is unpack the components, set them up, connect them with cables and turn them on. To do the job, you will need nothing more than a screwdriver and about half an hour of time.

Set up the system where you will be comfortable using it. Pick a place away from direct sunlight and heat: The computer needs to stay cool, and you need to avoid annoying glare on the monitor screen. Although you may sometimes want to work with the PCjr's unattached keyboard on your lap, you should still provide yourself with a table that is of a convenient height for typing and is large enough — at least 30 by 36 inches — to hold the system components, papers and a lamp. For your IBM manuals and other reference books, nearby shelves are useful.

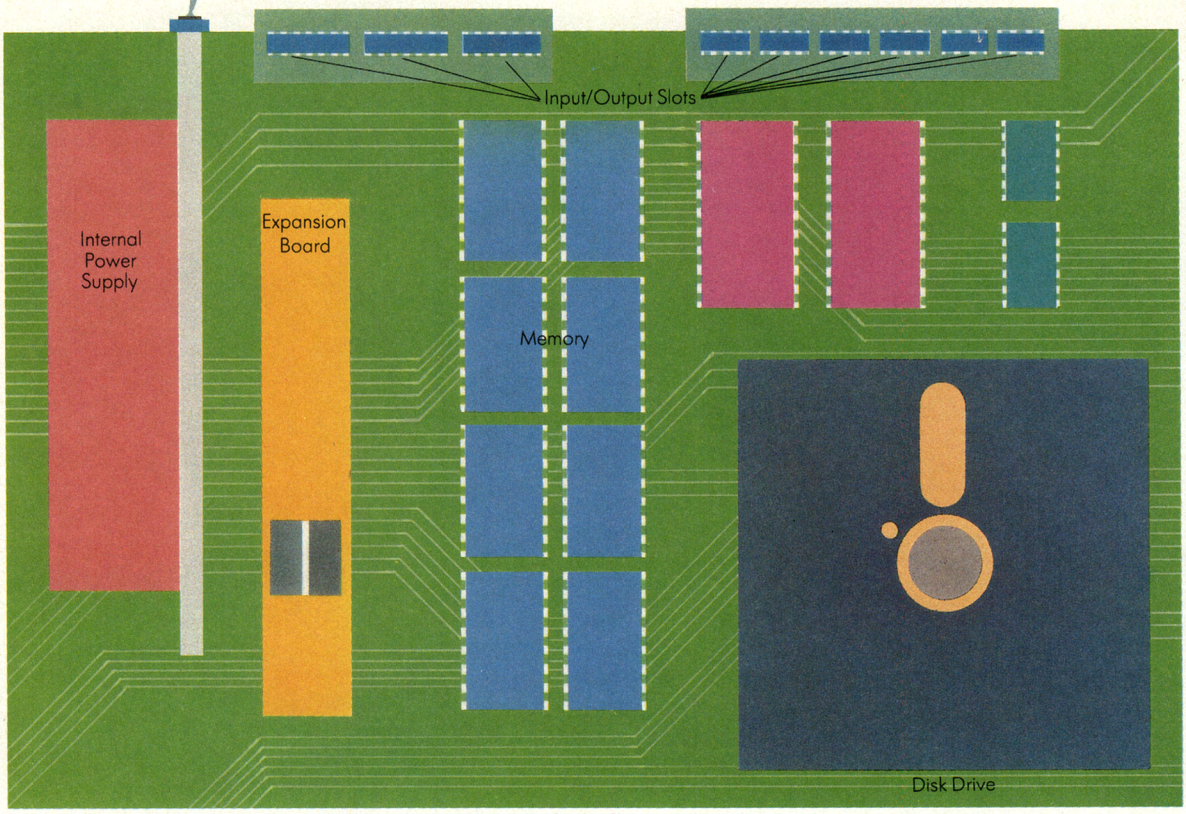
The system components are packed in separate compartments of one box, and they are protected by plastic bags and special foam inserts that prevent the build-up of static electricity, which could wreak havoc with the electronics of the system. Be sure to save all of these packing materials; you will need them to protect the equipment anytime you move it. And go carefully through the carton before you store it, making certain you have removed all printed matter.

The inside of each component is a marvel of electronic and mechanical complexity, but you don't need to know how a computer works before you start using it. Your PCjr already has the necessary connections for such optional equipment as a monitor, printer or game controllers. For the adapters these components may require, consult your dealer. And remember your dealer in case anything goes wrong as you set up the system; that's the reason you have a warranty. If anything doesn't work, retrace your steps and try again. If it still doesn't work, call the dealer.

The expanded PCjr system shown in this chapter includes a printer and an IBM RGB (color) monitor, in addition to the system unit with cordless remote keyboard, one disk drive and two cartridge slots. More specialized components such as communications devices and game attachments are discussed in later chapters. Whenever you have questions about installations, or hardware in general, you will find the IBM *Guide to Operations* — packaged with the computer — a useful reference.

Fresh from the box, an IBM PCjr keyboard sits atop its system unit. Weighing less than two pounds, this battery-operated keyboard can be held on your lap and operated with no cables attached. It sends signals to the system unit from up to 20 feet away, by means of infrared rays; you need only keep the back of the keyboard aimed roughly at the front of the computer.

At the Heart of the System

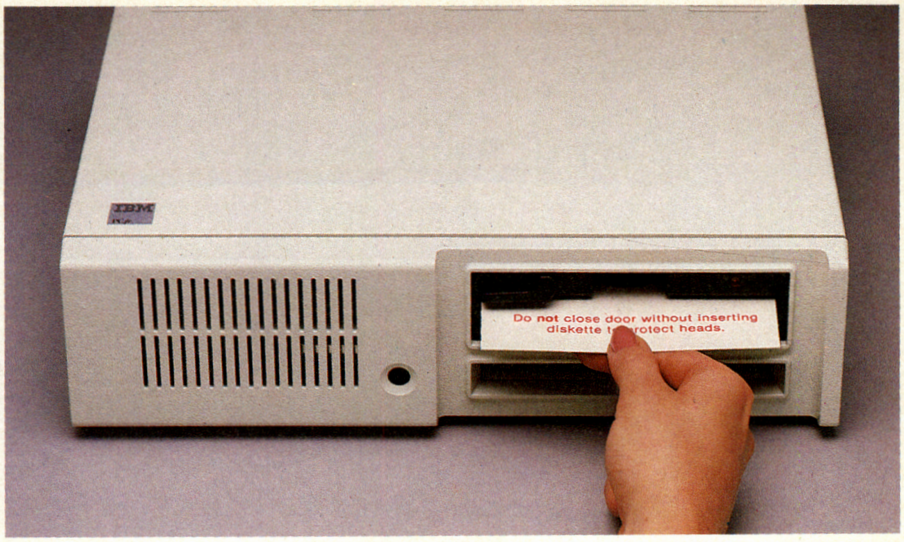


The system unit of the enhanced IBM PCjr, shown here schematically, houses the main processing and memory circuits, as well as the disk drive and the expansion board. The central processing unit, or CPU, which lies beneath the disk drive, processes and distributes information; memory stores it during processing, and the disk drive records it on floppy disks. The expansion board increases the PCjr's memory and allows display of 80 characters of text on each line.

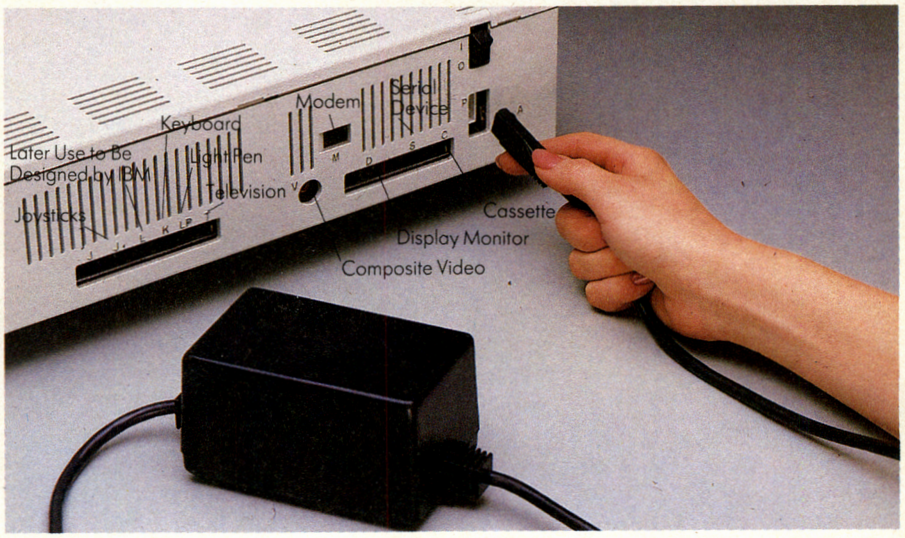
The system unit is the control center of the IBM PCjr. Inside this box are the circuits and mechanisms that process and store information and that send instructions to other parts of the system.

The computer is said to read when it takes information in, and to write when it sends information out. Much of the information passes to and from the computer's memory, which is housed in a set of specialized circuits. One part, called read-only memory (ROM), holds a permanent record of information the computer needs each time it is turned on. ROM cannot be added to or erased. Another part, the read-and-write, or random access, memory (RAM), lets the computer store and retrieve information. You can erase the contents of read-and-write memory to make room for new information; in fact, whenever the unit is switched off, RAM is wiped clean. For more permanent storage, the computer can also read and write information on floppy disks, which are inserted in the system unit's disk drive, or on magnetic tape.

A transformer, called the power supply, converts the household's 110-volt alternating current into the low-voltage direct current required by the PCjr's circuits. The power supply heats up when plugged in, so it is wise to unplug it from the outlet if you are not going to be using your computer for a few days.



The horizontal opening of the disk drive, located on the front of the system unit, may hold a protective cardboard insert when you unpack your system. You must remove the insert before operating the system; turn the latch at the left side of the opening and pull the cardboard out.



The back of the system unit has 13 input/output slots for cables that link the unit with other parts of the system. The power supply slot, labeled *P*, is at the right side of the back. Before you insert the rectangular plug of the power-supply cable into the system unit, make sure the switch above the connection is in the OFF position. Don't plug the other end of the power-supply cable into an outlet until you have connected the rest of the components to the system unit.

The PCjr's Two Models

There are two versions of the IBM PCjr — the entry model and the enhanced model. The entry model is the simpler and less expensive of the two. But there are significant similarities. The two models use identical keyboards, and both have two slots in the system unit for cartridges containing software. Each version has 64,000 bytes of read-only memory (a byte is the unit of memory required to store a single character).

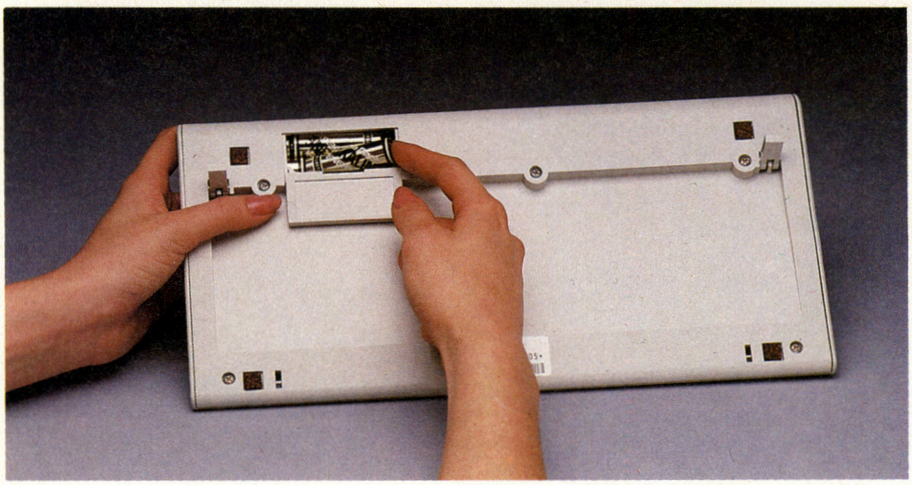
In other respects, however, the enhanced version is a more powerful and more versatile machine. It has 128,000 bytes of read-and-write memory, compared with the 64,000 of the entry model. The entry model lacks a built-in disk drive, relying instead on a cassette recorder for saving information. Each model can be used with either a home TV set or a monitor as a display device, but the entry model can display text only in the 40-column mode. You can add accessories to an entry-level machine to increase its usefulness; see pages 98-99.

The Keyboard: Your Connection to the PCjr

The IBM PCjr's keyboard resembles a typewriter's, except that it has additional keys. Keyboard input can be transmitted to the system unit either over a cable or by infrared signals emitted by diodes on the back of the keyboard. No matter which method of transmission you choose, you can use the keyboard in your lap or on a table.



When you are planning to rely on infrared signals for data transmission, turn the keyboard over and insert four AA alkaline batteries in the compartment on the bottom to provide power for the keyboard. To adjust the angle of the keyboard for tabletop use, fold out the two small plastic legs on the bottom of the keyboard.



The keyboard is the PCjr's main device for sending information to the computer. It looks like the front half of a typewriter with 62 keys, but striking a key yields a different kind of result. Instead of producing a symbol on a piece of paper, a keystroke sends a signal to the computer, where it is processed by the CPU.

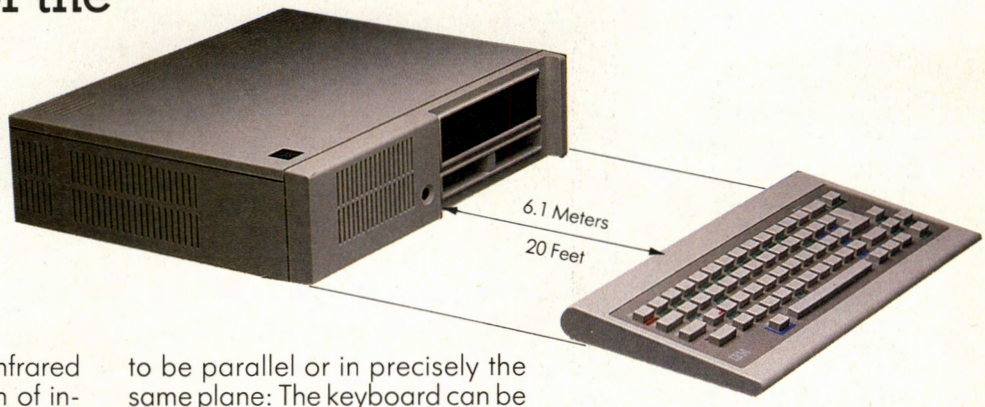
The keyboard has no ON/OFF switch. When not in use, it draws a negligible amount of current from its four AA batteries. But at the press of a key, the keyboard is immediately activated; it steps up the amount of current it draws and begins sending signals. When keystrokes cease, the keyboard becomes dormant again.

The computer's keyboard sends signals in one of two ways. On its back panel are two bulbs, known as light-emitting diodes (LEDs), that transmit signals via infrared light to the system unit. A photo-electric cell, which can be seen through a small opening on the front of the system unit, receives the infrared signals. Keyboard signals can also be transmitted to the system unit over a cable. You will need to use a cable set-up if strong sunlight disrupts reception of infrared signals, or if two IBM PCjrs operating in the same room interfere with each other's infrared signals. When you use the cable, the keyboard's batteries and LEDs automatically shut down and power is drawn from the system unit to run the keyboard.



To link the computer keyboard to the system unit with the cable, be sure that the unit's power switch is in the OFF position and unplug the power supply from the outlet bar or wall outlet. Push the cable terminal marked *K*, with the *K* facing up, into the connector labeled *K* on the back of the system unit. Push the cable's other terminal, tab side down, into the connector on the right end of the keyboard's back panel. In addition to transmitting input, the cable supplies current to the keyboard for its operation.

Positioning for the Infrared Link



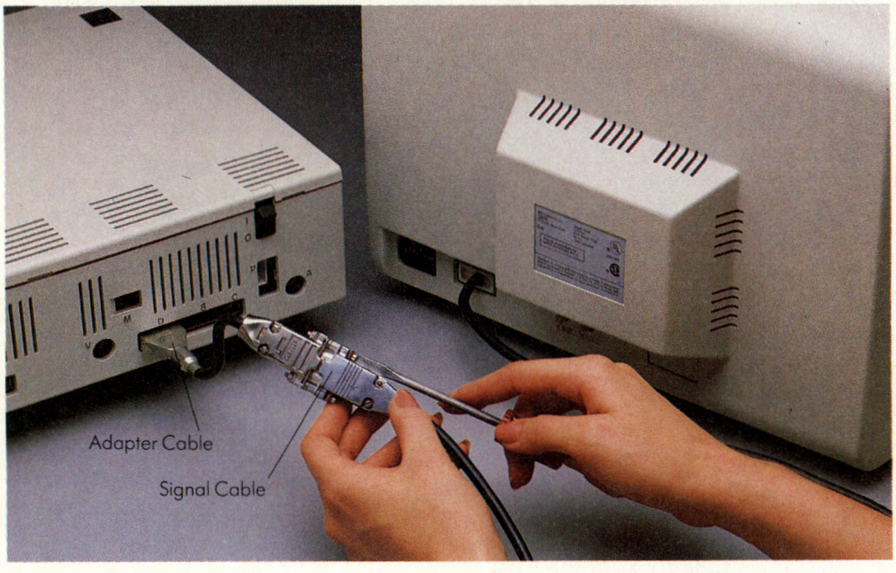
When you are relying on infrared signals for the transmission of input from the keyboard to the system unit, careful positioning is important. Even though the two components can be separated by as much as 20 feet, the back of the keyboard must still be oriented toward the front of the system unit. The two components do not need to be parallel or in precisely the same plane: The keyboard can be positioned to aim its signal anywhere within arcs of 60 degrees up or down from, or to the left or right of, dead center. Make sure that no solid object, such as a chair, lies in the line of transmission, since the infrared signals will not be able to penetrate it.

When a keyboard is out of range, or its signal is blocked or scrambled by too much direct sunlight, or if its signal power is reduced because of weak batteries, it will emit a beep to alert you to the faulty link.

Video Display: Several Options



An RGB monitor delivers high-intensity, high-resolution signals, an important factor if you plan to use software with detailed graphics. It displays an 80-character line, double that of most other color monitors. Some games software, however, will not display in color on RGB monitors.

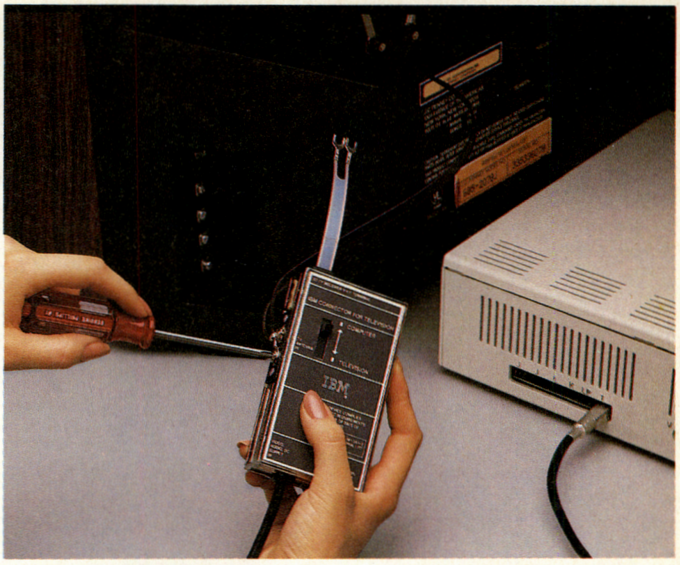


To connect the PCjr to an RGB monitor, turn off and unplug both units, then plug the end of the adapter cable marked *D*, label up, into the D connector on the back of the system unit. Insert the other end into the free end of the RGB monitor signal cable. Tighten the screws on either side of the connectors. Plug the color monitor's outlet cord into an electrical outlet.

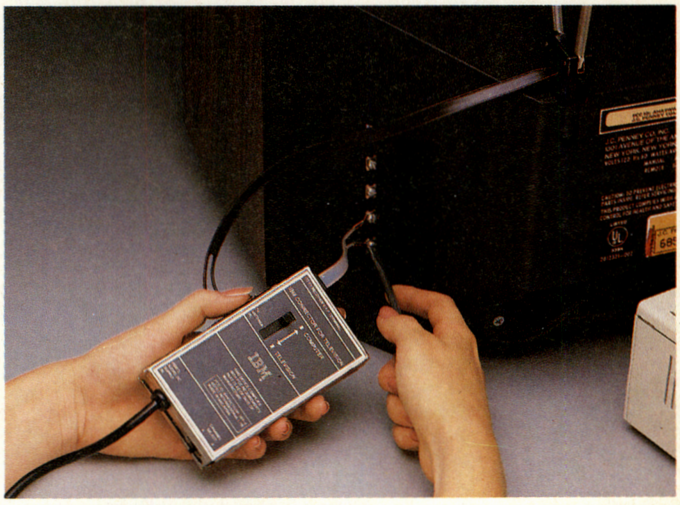
With a color monitor or color television set attached to your PCjr, the door to visually exciting software is wide open. If you decide to use your color television, you will need an RF (radio frequency) modulator, to change the computer signals to a form the TV can accept. Using a television as a monitor, you can write lines 40 characters long and display graphics in color, but the color quality will depend on the quality of your set. With some television screens, letters can be very difficult to read.

For better clarity and color intensity, you might choose between two types of color monitor, the composite video and the RGB monitor. A composite-video monitor uses a single signal to control the three color "guns" behind the screen, the same as a televison but with a higher resolution of color. Composite-video monitors use a single-pin phono jack. With the RGB (for red, green and blue), the computer sends a separate signal for each of the three colors, providing the greatest resolution and intensity. In addition, the screen accommodates lines 80 characters long, an advantage if you plan to use the computer as a word processor. An IBM adapter cable is necessary to link the RGB monitor with the PCjr.

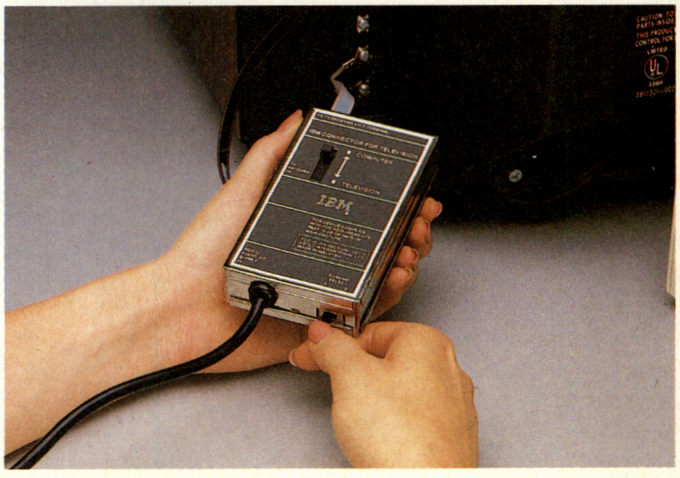
Linking the PCjr to Your Television Set



To connect a TV set to the PCjr, turn both off and unplug them from household current. Plug the cord marked *T* on the IBM Connector for Television into the socket labeled *T* on the back of the system unit; the *T* on the IBM connector's plug must face up. With a screwdriver, loosen the screws labeled VHF on the back of the TV set to free the twin leads of the VHF antenna. Slip these antenna leads under the two screws on the side of the IBM connector; tighten the screws.

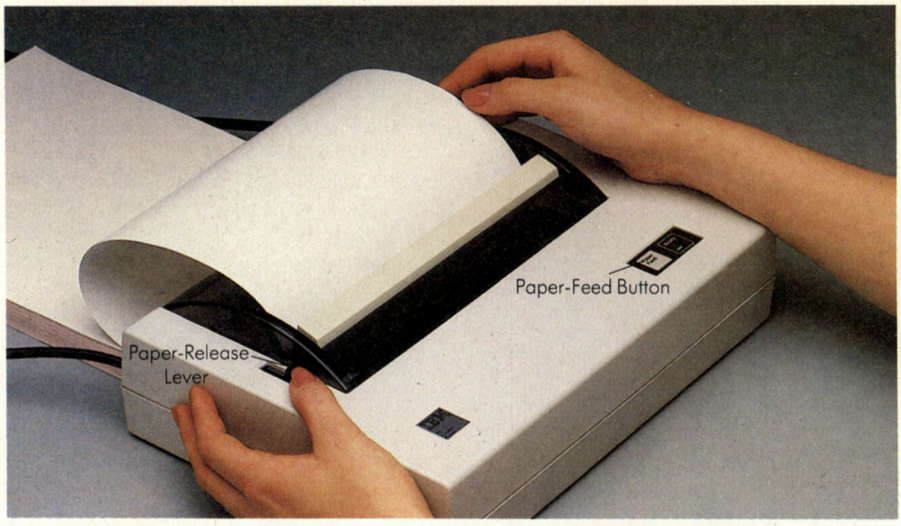


An antenna wire labeled TO TV RECEIVER VHF TERMINAL extends from one end of the IBM Connector for Television. Slip the twin leads of the antenna wire under the television set's VHF screws; use a Phillips screwdriver to tighten the screws *(above)*.

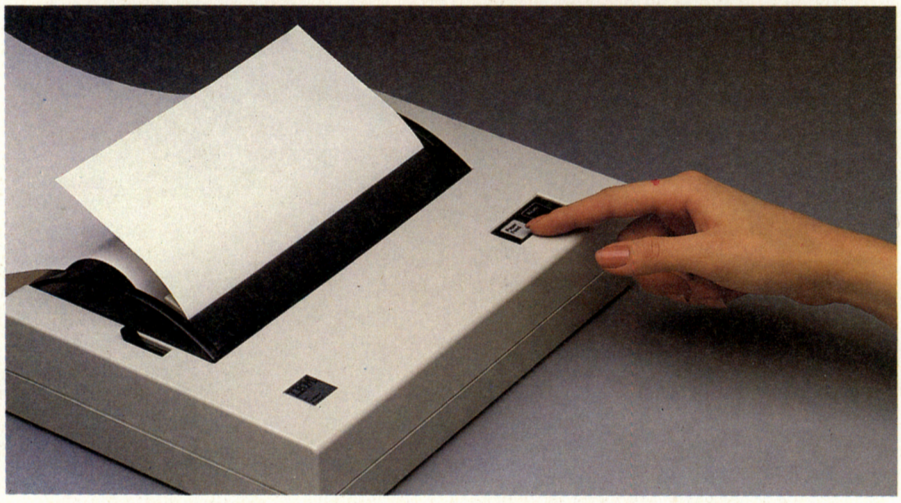


To use the TV set for computer display, set the IBM connector's channel-select switch for channel 4 or channel 3, whichever one is not broadcast in your area, set the switch on the connector's face on COMPUTER, and turn the PCjr on. To return to television viewing, set the switch on the IBM connector's face to TELEVISION.

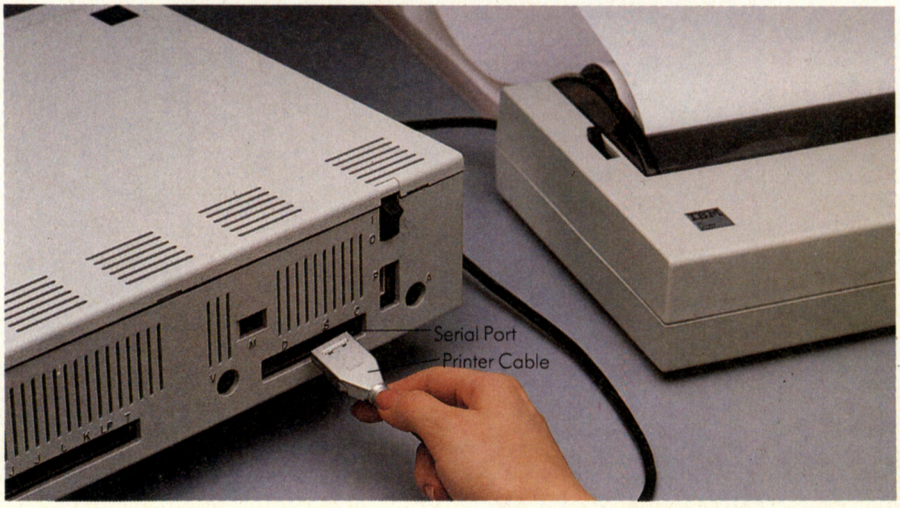
Setting Up the Thermal Printer



Once the printer is plugged in and turned on (the ready signal light will glow), pull the paper-release lever forward and, with the plastic access cover closed, insert the leading edge of a stack of fanfold paper through the slot in the top of the cover. Push the paper into the paper path until it stops. Lock it in place by returning the paper-release lever to its original position.



Press and hold down the paper-feed button to advance the paper. Stop when the paper is positioned properly for your first line. To be certain that paper is correctly loaded and the printer is operating, run a printer self-test. Switch off the power and press and hold down the paper feed; switch the power on and when the machine starts printing lines of letters, numbers and symbols, release the feed button. Stop the test by turning off the machine.



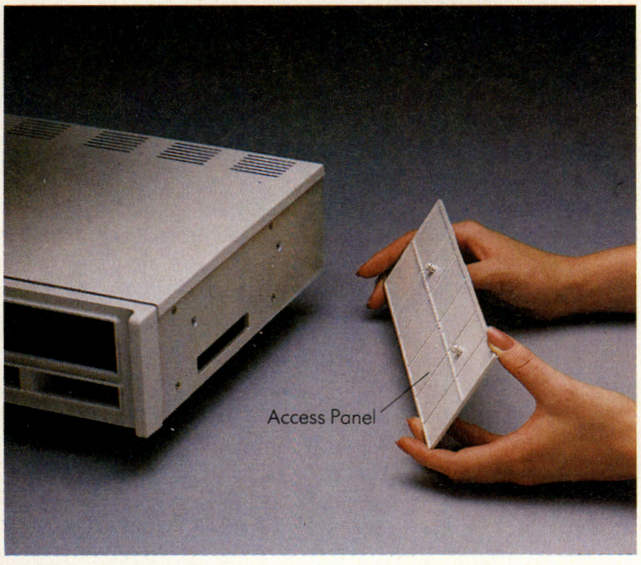
With the power switches of both printer and computer turned off, plug the printer cable into the serial port, which is labeled *S*, at the rear of the computer. Before using the printer, turn on the power first to the computer, then to the printer.

Parallel Port: Gateway to Expansion

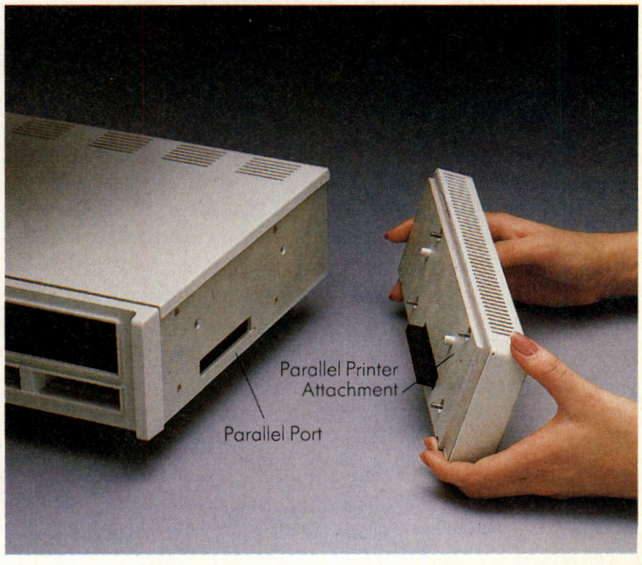
The Compact Printer isn't the only printer you can attach to the PCjr's serial port. With an adapter cable, you can use letter-quality or daisy-wheel printers that produce type like that from a typewriter.

However, if speed and flexibility are more important to you, you may decide on a dot-matrix printer. Dot-matrix printers print each character as a set of dots, and use standard fanfold paper; some can also print on single sheets. They are very fast, printing 80 to 300 characters per second, or about 960 to 3,600 words per minute. Some parallel printers are also designed to print graphics as well as characters. You may even want a color printer.

A dot-matrix or color printer must be connected via the optional Parallel Printer Attachment. Remove the PCjr's right side panel *(below)*, secure the attachment in place, and cover it with the side panel. A cable from the printer then plugs into the rear of the attachment.



To remove the access panel on the system unit, first turn off the unit and unplug the power cord. Grip the slot at the base of the panel on the right side and pull the panel off. You may need to pry it gently with a nonmagnetized flathead screwdriver.



Hold the Parallel Printer Attachment so as to align its mounting pins with mounting holes on the side of the system unit, and to align the attachment connector with the parallel-port slot. Press the attachment into place and tighten the screws. Affix the access panel to the attachment by snapping it into place.

By connecting a printer to the PCjr, you can preserve the work of your computer, a convenience in many instances and essential to utilizing the PCjr's word processing capabilities.

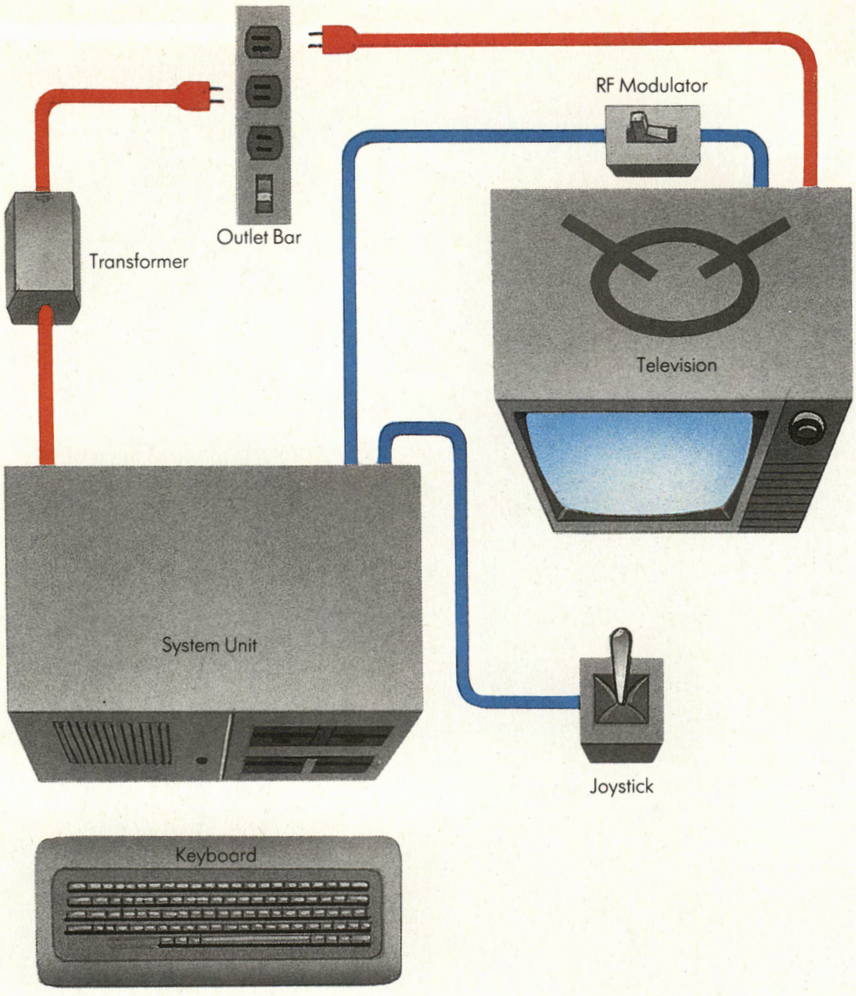
The IBM printer designed as a companion to the PCjr is the lightweight (less than seven pounds), low-cost Compact Printer. Rather than imprinting ink on paper, the Compact Printer uses hundreds of tiny heated pins to burn impressions onto thermal paper. The printer can reproduce anything visible on the monitor, including graphics, and characters in three type sizes.

Thermal paper is available in single sheets, in a roll that fits into the printer, or in fanfold, a length of folded, perforated paper that sits behind or below the printer and feeds into it.

Yet the Compact Printer has its drawbacks. Thermal paper darkens with age or exposure to sunlight, so the information printed on it will eventually be lost. It prints in only one direction, from left to right, and at a rate of about 50 characters per second, which is slower than many other printers. And while the characters are legible, they do not have the look of letter-quality printers.

If these are critical considerations for your needs, you can connect other printer models to the PCjr's serial port by using an optional adapter cable or by affixing to the computer a special interface known as a parallel printer adapter *(above)*.

Putting It All Together as a System



A PCjr used for family games and education utilizes an entry-level system unit, keyboard, transformer, RF modulator, joystick, TV and outlet bar. Cables shown in red carry the electricity for powering each component. The transformer converts the household's 110-volt alternating current to the low-voltage direct current used by the computer. Other cables *(blue)* convey the electrical impulses that encode information to be processed by the components.

Linking the components of your PCjr is simple. You need only position the components conveniently and remember a few precautions.

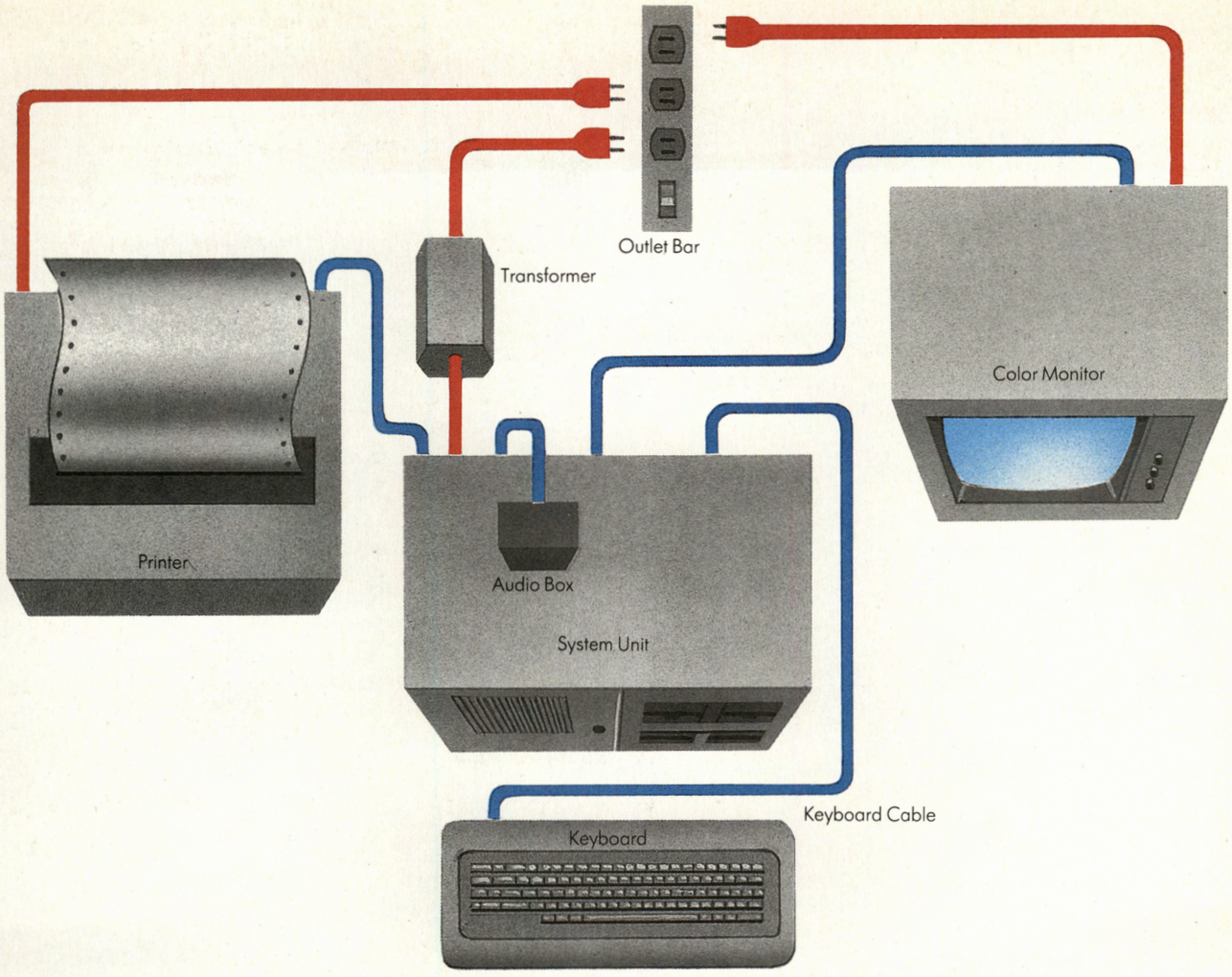
Place the system unit on a table and the keyboard in front of it. See that nothing blocks the infrared signal that links the keyboard and the system unit. Place the television or video monitor where you can comfortably see the screen. A TV or monitor should be at least 6 inches from a disk drive, however; it can generate signals that interfere with the recording of information.

You can put the printer anywhere up off the floor. If you use an outlet bar *(box, far right)* put it on the floor near a wall, out of the way of accidental kicks.

Be sure not to block any component's fan vents. And leave room, away from the computer, for such things as coffee cups — spills can cause extensive damage.

The cables that link the components contain multiple wires. The power cables that draw household current have three wires and three-prong plugs. Data cables, which carry data encoded in electrical impulses to the components, have as many as 25 wires, and connectors to match. These cables can be damaged if bent sharply or pinched.

If a connector doesn't plug in easily, turn it over; most have only one correct orientation. If it still doesn't fit, you may be trying the wrong socket.



For the home office, a fully complemented PCjr includes an enhanced system unit with a disk drive, color monitor, audio box, printer, transformer, outlet bar and keyboard connected by cable to the system unit. Cables shown here in red conduct power; those shown in blue transmit data.

Picking Your Power Source

The best way to plug in your system is with an outlet bar — several outlets controlled by one switch *(right)*. Get one that has a built-in regulator, which will protect the system from power surges.

Don't get power from just any wall outlet: Only grounded (three-hole) outlets protect the computer's circuits properly. Avoid jury-rigged arrangements with extension cords or three-way adapters.

Avoid sharing circuits with large appliances such as refrigerators, which often drain current when they switch on. And don't use the system during electrical storms, which can cause voltage surges that could burn out the computer.



An outlet bar provides a separate circuit for your system. A built-in regulator absorbs abnormal surges of electricity, and a single switch controls all the outlets.


128

Turning On the Machine and Getting Started

With all its components in place and hooked up, your PCjr can go to work as soon as you flick the power switch: Contained in the computer's read-only memory (ROM) are information and instructions that make it instantly functional. Each time you turn the PCjr on, it performs a wake-up routine called the power-up self-test, a brief automatic program that checks out the system and any IBM attachments. While this test is running, the IBM logo and color bar appear on the monitor *(left)*.

At your command, another set of instructions, the diagnostic routines *(pages 24-25)*, can direct the machine to conduct a more extensive test of all components. There is also a built-in tutorial called Keyboard Adventure, which you can call up for an introduction to the machine and its keyboard *(page 23)*.

PCjr's keyboard is the primary device for communicating with the computer and with peripheral components such as printers. It resembles the keyboard of a conventional typewriter, with several special keys added. The alphabetic, numeric and punctuation keys function much as they do on a regular typewriter, but the special keys have functions peculiar to a computer. Some of the special keys control the movement of the short-flashing marker called the cursor. Others instruct the computer to enter information in its memory or to erase a character. Still others, in combination, are the equivalent of a panic button: Pressing them tells the computer to stop whatever it is doing and wait for further instructions.

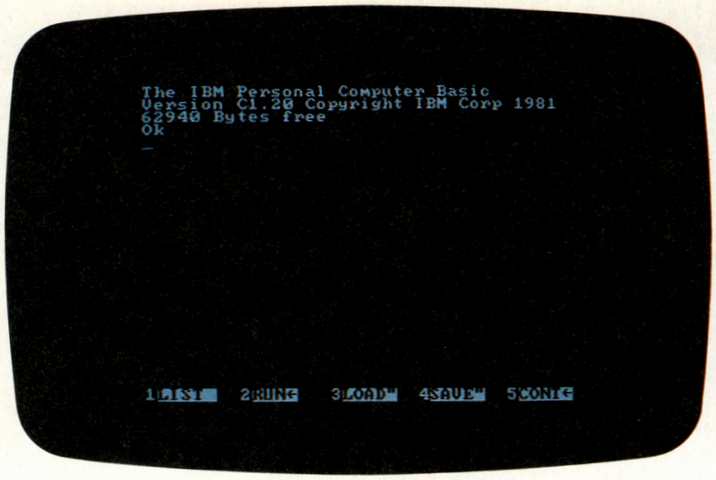
Also built into PCjr are color, graphics and sound capabilities. Given the right commands, the machine can generate up to 16 foreground and four background colors. From a set of dot patterns, it can create graphics on the video screen. It can produce all 12 notes in each of the seven octaves of a piano keyboard — including sharps and flats. It can also play three-note chords and any tempo you select.

Finally, PCjr comes with a built-in language called BASIC *(pages 26-27)*, which you can use to communicate in English with the machine. Cartridge BASIC *(pages 28-29)*, a more advanced version that you plug into one of the machine's cartridge slots, offers more commands and capabilities. Learning the language takes practice, but once you have mastered it, you will be able to program the machine to do your bidding *(Chapter 7)*.

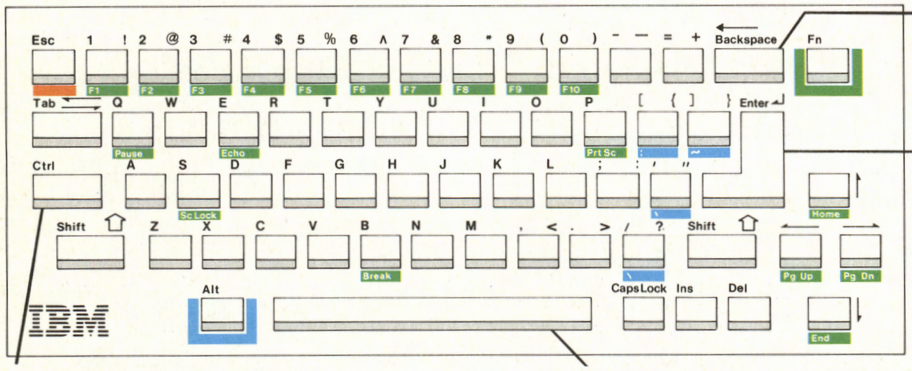
When the PCjr is up and running, the first image to appear on the monitor is the IBM logo and a color bar. The monitor, a cordless keyboard and a system unit with disk drive and twin cartridge slots form the core of an enhanced PCjr system.

Warming Up with Keys and Screen

Turn on the system by switching on first the outlet bar and then the individual components, leaving the system unit for last. When a message like the one at right appears, the PCjr is ready for instructions. If the writing is off center, hold down the **Ctrl** and **Alt** keys and use the cursor-control keys *(chart, opposite)*, to shift the message left or right.



Keys in the central portion of the keyboard behave like their counterparts on a typewriter. Keys marked with a green or blue stripe carry out additional tasks when used with a corresponding color-coded key: Green-coded keys may be used with the **Function** (**Fn**) key *(far right)*, blue-coded keys with the **Alternate** (**Alt**) key.



The **Backspace** key (←) is used when you need to make a correction. It moves the cursor one space to the left, erasing the letter there to make room for a new one.

The **Enter** key (↵) instructs the computer to begin processing the line of information or instruction you have just typed.

The **Control** key (**Ctrl**) changes the meanings of other keys when it is held down as they are pressed. It is used for a number of special functions *(chart, opposite)*.

The space bar deletes characters as it moves the cursor forward.

If you operate the PCjr without a disk or a cartridge, it uses a computer language called BASIC, which is built into its circuits. When the computer is ready, it writes a message in BASIC on the screen *(above)*. The message tells you how much memory is available to process information, expressed as the number of bytes free. (A byte is a unit of memory that holds one character.) The word *Ok* just above the cursor is a prompt in BASIC that means the computer is ready for your next command; other languages use other prompts. Ignore the line at the bottom of the screen for now; it lists some special BASIC commands *(Chapter 7)*.

The computer is ready, so explore the keyboard. Be fearless: You can't hurt the machine by pushing keys. Note the difference between the letters *l* and *O* and the numerals *1* and *0;* the computer will not accept substitutions. If the message "syntax error" appears, it simply means the computer doesn't understand your instructions. Just keep going. If the PCjr doesn't react to anything you do, turn it off, give it 20 seconds to clear its circuits and turn it on again.

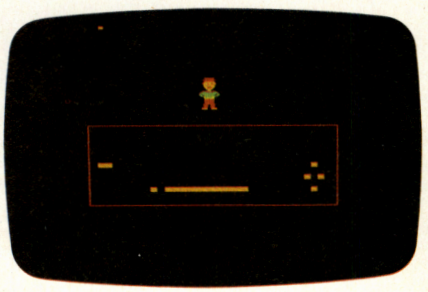
The function of each key on the PCjr's keyboard depends on the set of instructions, or program, in control of the machine. Some programs include paper overlays to relabel the keys; you can also purchase blank overlays to assign functions for programs you write yourself.

How Special Keys Function in BASIC

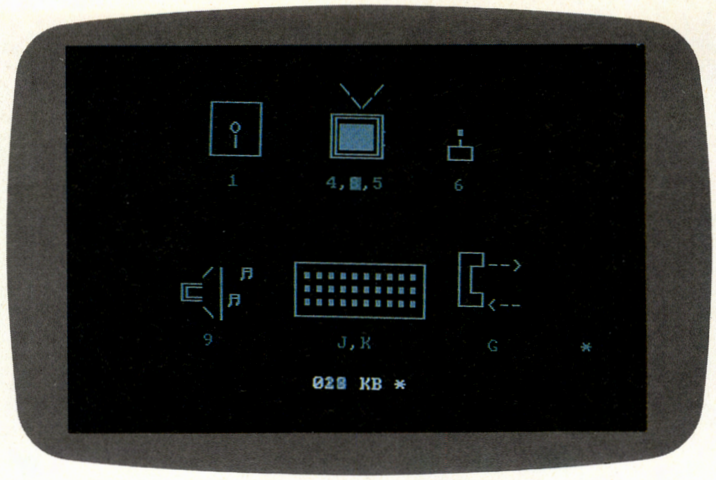
← ↑ → ↓	These four keys, grouped on the right side of the keyboard, move the cursor one space at a time in the direction of the arrows. Holding a key down makes the cursor move until the key is released.
Ctrl + Fn/Home	Holding down **Ctrl** and pressing **Fn** and **Home** clears the screen and puts the cursor in the top left corner of the screen.
Ctrl + Fn/End	This combination erases characters, from the cursor position to the end of a typed line. All lines on the screen are erased until the last **Enter** is found.
Ctrl + →	Holding down **Ctrl** and pressing the → key moves the cursor to the right until it reaches a new word.
Ctrl + ←	This combination moves the cursor left to the end of the previous word.
Ins Ctrl + R	These keys are on/off toggles for the **insert** mode. Pressing **Ins** alone or **Ctrl** plus **R** causes the computer to enlarge the cursor and make room for an insertion to the left of it. Pressing again turns **insert** off.
Del Ctrl + H ←	All three sets of keys delete individual characters. Use **Del** to erase the character in the cursor position; use either **Backspace** alone or **Ctrl** and **H** together to erase the character to the left of the cursor.
Esc	Pressing **Escape** (**Esc**) erases from the screen all information on the line where the cursor is located.
Fn + Break	By holding down the **Fn** key while pressing **Break**, you stop any process the computer is performing. The computer then awaits your next instruction.
Fn + Pause	Pressing **Fn** and the **Pause** key puts the computer in a suspended state. These keys temporarily halt printing or program listing. To end this state, press any key other than **Shift**, **Ins** or **Fn/Break**.
Fn + PrtSC	By holding the **Fn** key and pressing **PrtSc**, you instruct the computer to print everything on the screen. Be sure the printer is on and ready to print.
Fn + Echo	Pressing **Fn** and **Echo** serves as an on/off toggle switch that allows text sent to the screen to be sent to the printer at the same time. Pressing both keys again stops the print echo.

Adventure on the Keyboard

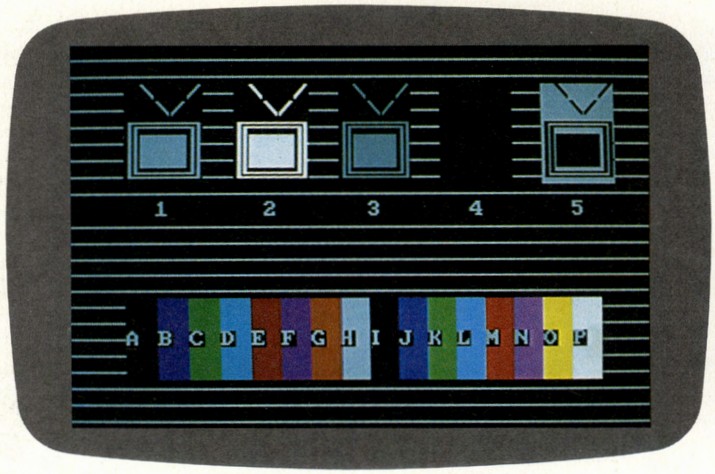
When the message on the screen opposite is on your screen, pressing the **Escape** (**Esc**) key calls up a tutorial program — *Keyboard Adventure* — to help acquaint you with the PCjr's keyboard. (The Adventure is described in your *Guide to Operations*, starting on page 2.) The program uses print, graphics, sound and a cartoon guide named P.C. as learning aids. When you press a key, P.C. puts a symbol for that key in the appropriate spot on the screen keyboard. P.C. also demonstrates the functions of special keys and key combinations. To leave the Adventure, hold down **Ctrl** and **Atl** and press **Del**.



Checking the System with Built-in Tests



To clear the PCjr's circuits to begin the diagnostic tests, reset the system by holding down **Ctrl** and **Alt** and pressing **Del**. When the *Ok* prompt appears, hold down **Ctrl** and **Alt** and press **Ins** to call up a Test Menu, or set of options *(right)*. The icons here represent system components to be tested. Press **Ins** to move the cursor to the Test Tag, or character, under the icon you want and press **Enter** ↵ to begin the test. Follow along in the Testing chapter of your PCjr *Guide to Operations*.



To test a monitor with 80-column display, select Test Tag 8 and turn to the *Guide* when the screen at right appears. Icon No. 3 should be blinking. Check the shapes of the blocks and figures on your screen against those in the *Guide to Operations;* if any do not match, the *Guide* explains what to do. To continue through the 19 remaining screens of this test, press **Ins** to return to the first Test Menu, press **Fn** and then **Break**.

Whenever you set up, move or add a component to your PCjr, you should run the series of self-tests built into the machine's circuits. The tests can check only standard IBM components, however, so before beginning you should remove all non-IBM attachments and cables except the display or TV. During the tests you will need to refer to the Testing chapter in the PCjr *Guide to Operations*.

The six icons, or figures, that appear on the Test Menu screen *(top of page)* represent peripherals such as the keyboard or display, or expansion options such as a light pen or a modem. Each icon is labeled by a character called a Test Tag. Test Tags for options will blink only if the option has been installed.

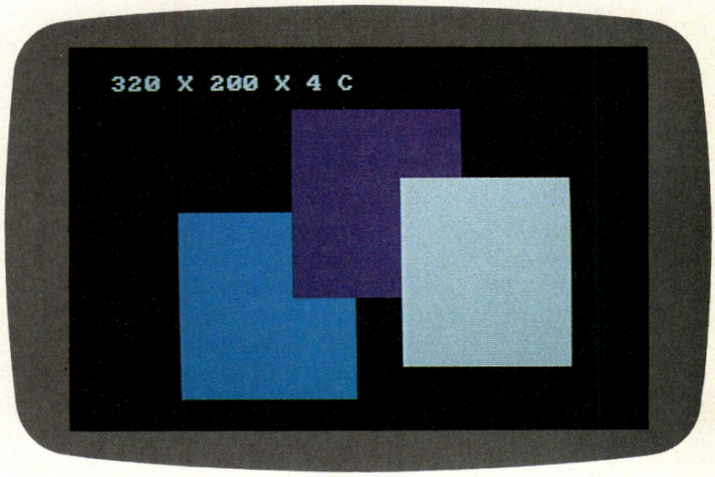
The test for each component usually has several parts. Simply compare the shapes and words that appear on your display with the test screens in the *Guide* (the colors on your display may differ from those in the booklet). The *Guide* advises you whether you can solve a problem yourself or should see a dealer.

When all parts of one test have been completed successfully, an asterisk will appear beneath that Test Tag and the cursor will blink at the next tag. You can choose another tag by pressing **Ins**, to move the cursor, and **Enter** to start the test. If you press **Ins** when the cursor is on the last tag on the first Test Menu, a second menu will appear.

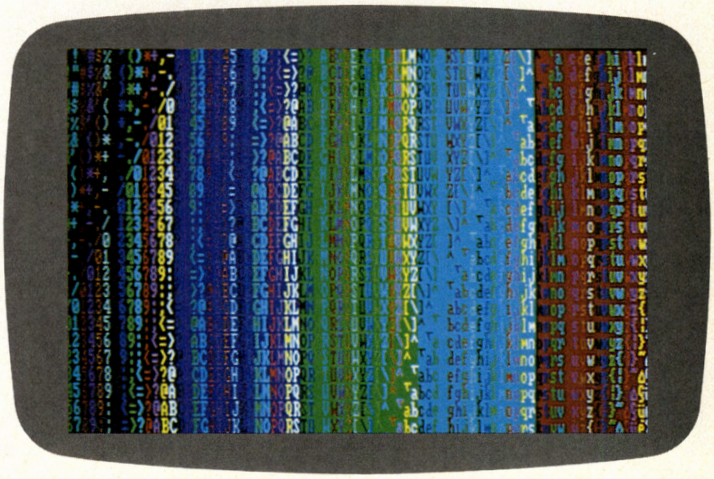
The screen at right is called the ASCII (pronounced AS-Key) screen. It is the second screen that appears in the 80-Column Display Test. Check the symbols and pattern shown on your screen against the illustration in the *Guide to Operations*. The symbols represent commands that the PCjr uses internally to store and communicate characters; you would not ordinarily see them. Press **Ins** to continue the test.



The sixth screen to appear during the test of an 80-column display reveals three interlocking blocks of color *(right)*. Check to see that the shapes match those on the corresponding screen in the *Guide;* it does not matter if the colors differ. Press **Ins** to continue with the rest of the test.



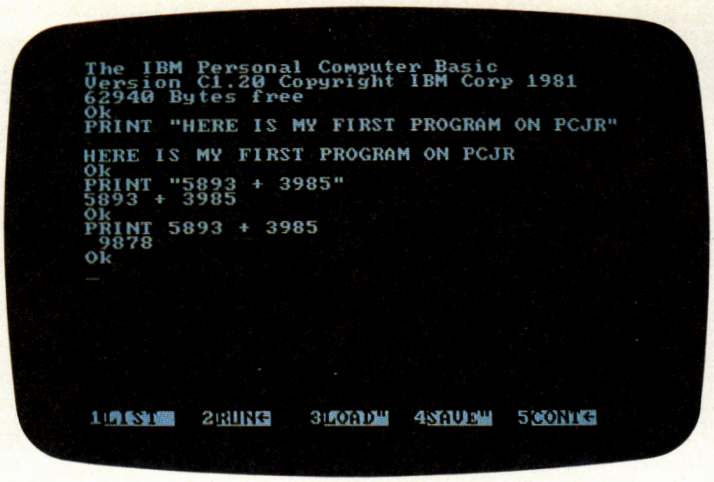
The ninth screen in the series displays the keyboard characters in vertical bands of color. Compare the shapes of the blocks and figures on your screen with those in the *Guide* (again, colors may differ) and press **Ins** to continue. When all the screens of this test have run, the first Test Menu screen will reappear. You may continue testing other components if you wish. To leave the diagnostics mode and reset the system, hold down **Ctrl** and **Alt** and press **Del**.



PCjr's Language: a Taste of BASIC

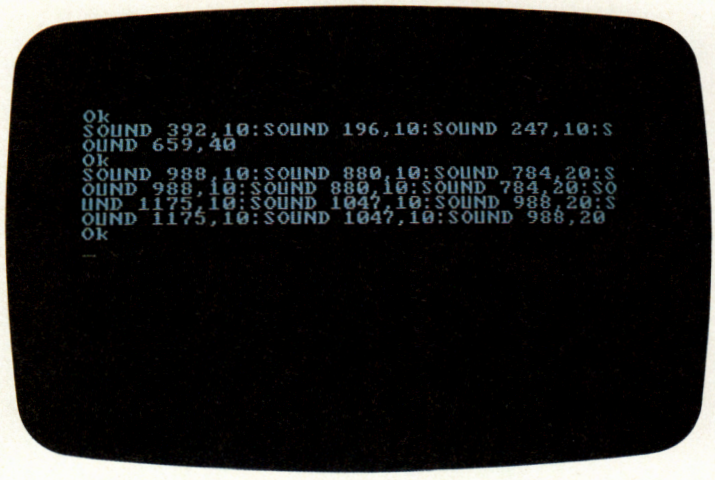
PRINTING AND CALCULATION

For an introduction to the powers of the PCjr's resident language, hold down **Ctrl** and **Alt** and press **Del** to reset the system. When the *Ok* prompt appears, type the command statements exactly as they appear at right, starting with the **print** command. Press **Enter** at the end of each line. Note the different result caused by omitting the quotation marks in the last statement.



MUSIC AND SOUND

Clear the screen and turn off the key indicators at the bottom of the screen by typing **cls:key off** and pressing **Enter**. Now type the command statement, beginning with the word **sound**. Press **Enter**. You should hear the four notes of the ukulele tuning, "My dog has fleas." If not, check for typing errors and try again. To play another tune, type the last statement carefully and press **Enter**.



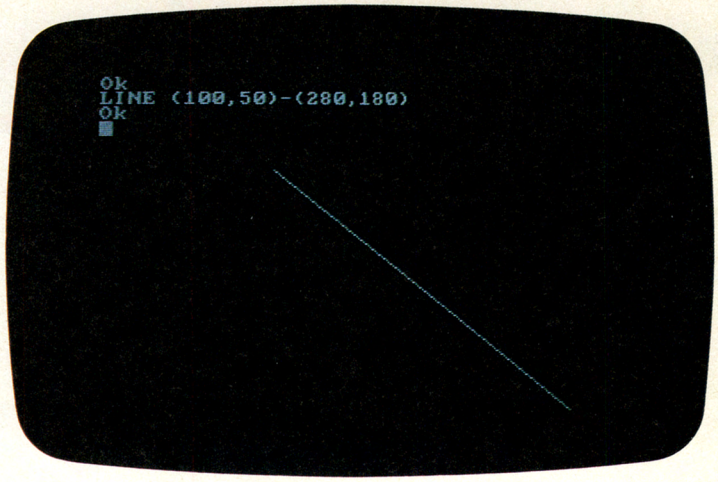
BASIC, the computer language you use to communicate with the PCjr, is an integral part of the machine. When you turn the power on, the PCjr wakes up in Cassette BASIC, a special form of the language residing in the computer's read-only memory (ROM). A more advanced form is Cartridge BASIC, which comes in a plastic cartridge, about the size of a deck of cards, that plugs into a slot in the front of the unit.

Cassette BASIC is so named because it includes commands that let you transfer information and programs to an audio-cassette tape recorder for storage on magnetic tape, in much the same way you can record, or store, music or voice. If your PCjr does not have a disk drive, you must use a cassette recorder to store your work; otherwise it will be lost when you turn the computer off. Cassette BASIC also offers numeric functions such as addition, subtraction, multiplication, division and square root, as well as the ability to handle text and to produce color graphics.

Cartridge BASIC, with 32K of additional ROM, is an optional purchase designed to extend the capabilities offered by Cassette BASIC. It adds commands for using the disk drive to store and retrieve information, as well as powerful new commands for video graphics and sound. Cartridge BASIC also allows the PCjr to operate with a modem for telecommunications *(Chapter 6)*, with

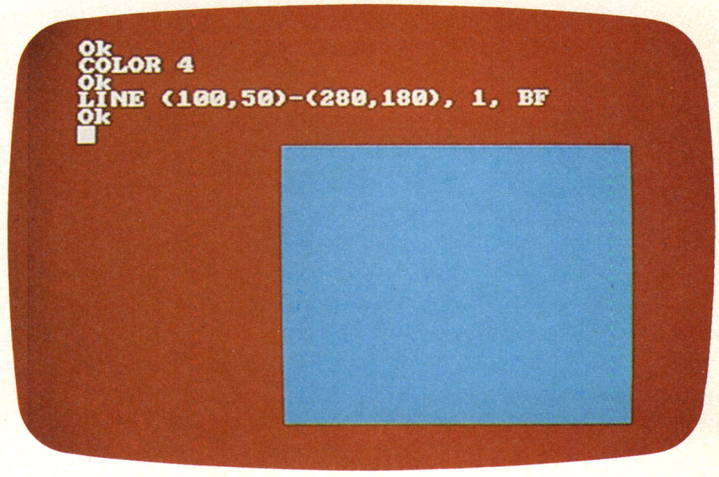
LINE GRAPHICS

To execute graphics commands, you must first switch from the text screen you have been using to a graphics screen. Type **SCREEN 1**, and press **Enter**. Now type the command statement shown on the screen at right exactly as it appears, and press **Enter**. A diagonal line should appear on the display. If it does not, check for typing errors and try typing the statement again.



COLOR GRAPHICS

Clear the screen by typing **CLS** and pressing **Enter**. Next, change the screen's background color by typing **COLOR 4** and pressing **Enter**. Type and enter the second command statement shown on the screen at right exactly as it appears. A light-blue square should appear on your display. If it does not, check for typing errors and try again.



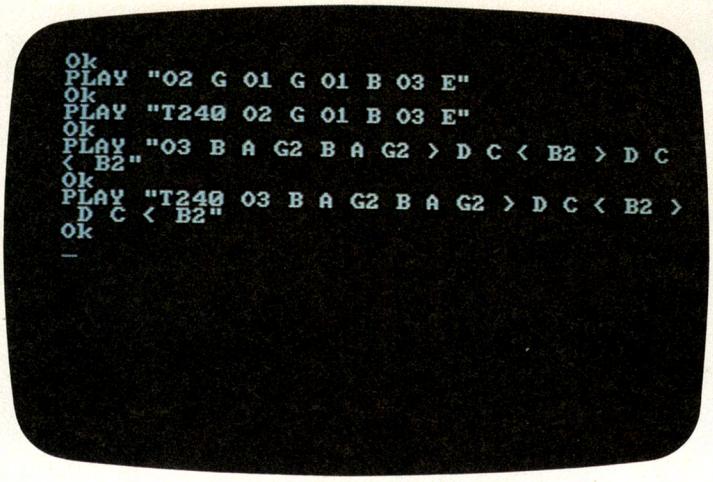
a printer for producing a hard copy of your work, or with other equipment such as joysticks and light pens.

With Cassette BASIC as a building block, Cartridge BASIC extends the total number of available commands, statements and functions to more than 200. Because all this additional power is kept on a ROM cartridge, the PCjr's read-and-write memory is left free for any work you want to give it.

The routines and screens shown here and on the next two pages are intended to give you a quick tour of some of the highlights of Cassette and Cartridge BASIC. Note the differences in how the two levels of BASIC handle typed text or statements, perform mathematical functions, produce a sound or a tune and create lines and geometric figures.

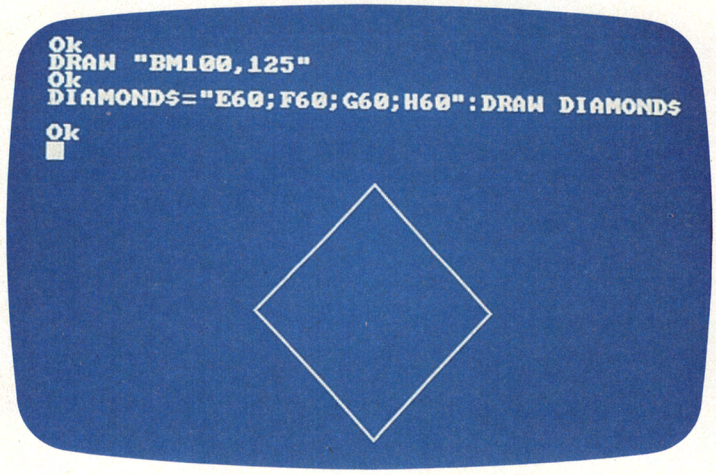
Although all of these routines are actually computer programs in simple form, you need not try to decipher and understand them here; in Chapter 7 you will be more formally introduced to BASIC programs. For now, you need only type exactly what is shown on each screen, then watch and listen to what the PCjr does in response to your commands. If you make a typing error, just backspace to erase it. The examples here are merely a hint of the kinds of things it is possible to do with Cassette BASIC and Cartridge BASIC.

The Expanded Power of Cartridge BASIC



MUSIC

First insert Cartridge BASIC *(box, below)* and turn on the machine. Clear the screen by typing **cls:key off**. Type and enter the commands exactly as shown on the screen at right. Be sure to type the letter *O* and not the numeral *0*. You should first hear the four notes of "My dog has fleas," just as you heard them in the Cassette BASIC example on page 26. Then you should hear the notes again at a faster tempo. Last comes the opening of "Three Blind Mice."



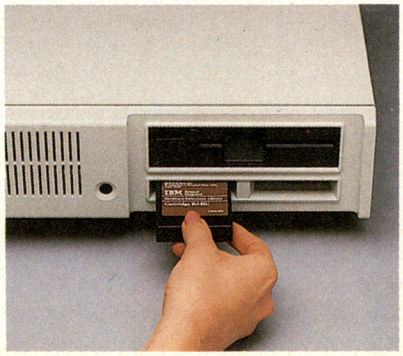
LINE GRAPHICS

Type **screen 1:color 9,1** and press **Enter**. The monitor should now display white text on a light-blue background, with only the *Ok* prompt and the cursor visible. Type each of the statements exactly as shown on the screen at right, pressing **Enter** after each statement. Your screen should now display a diamond. If you are using a monitor other than an IBM or a TV, the colors you see may differ from those shown here.

Plugging In a Cartridge

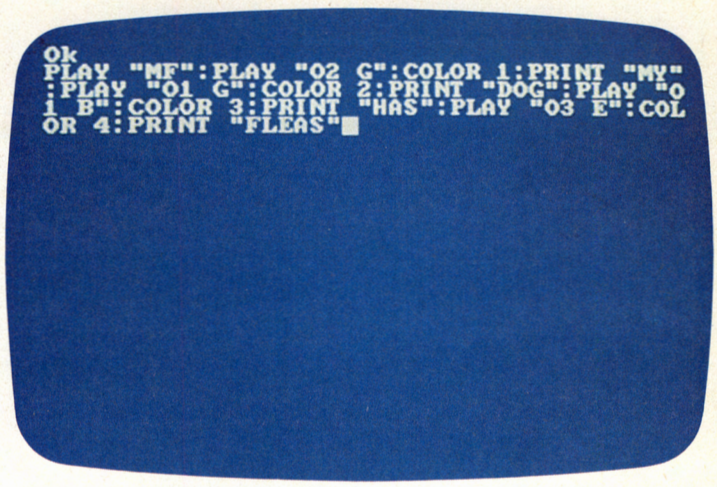
It does not matter whether the power is on or off when you insert a cartridge into the PCjr's system unit; neither the cartridge nor the computer will be damaged. But remember that if you plug in or remove a cartridge while the machine is on, the PCjr will automatically reset itself; anything on the screen or in memory will be lost.

To insert a cartridge, hold it label side up, the open end with its protruding circuit card pointed away from you. Push the cartridge firmly into one of the slots on the right side of the system unit's front panel until it snaps into place. To remove a cartridge, grip it firmly and pull.



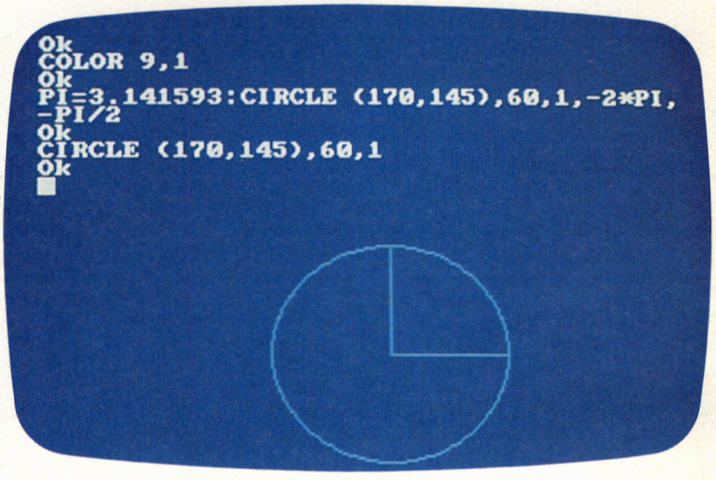
COMBINING MUSIC, PRINT AND COLOR

Type **cls** and press **Enter** to clear the screen. Type the command statement as it appears on the screen at right; pay close attention to the position of quotation marks and colons. Press **Enter**, then watch and listen. The words MY DOG HAS FLEAS should appear and the screen's background color should change as the note for each word is played. If not, check for typing errors, clear the screen and try again.



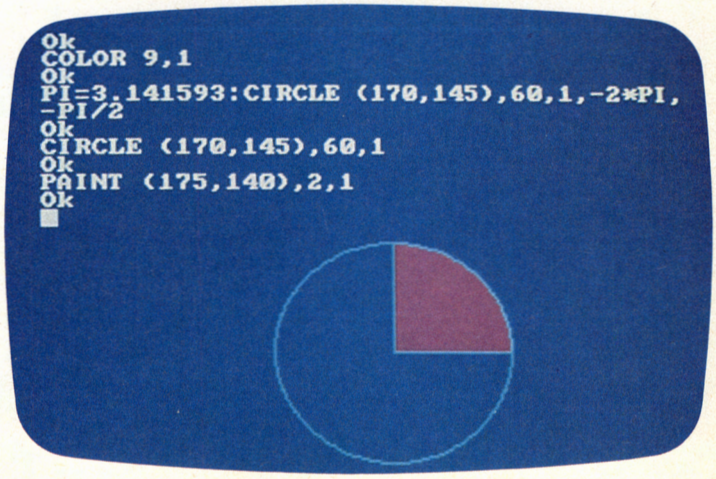
CIRCLES

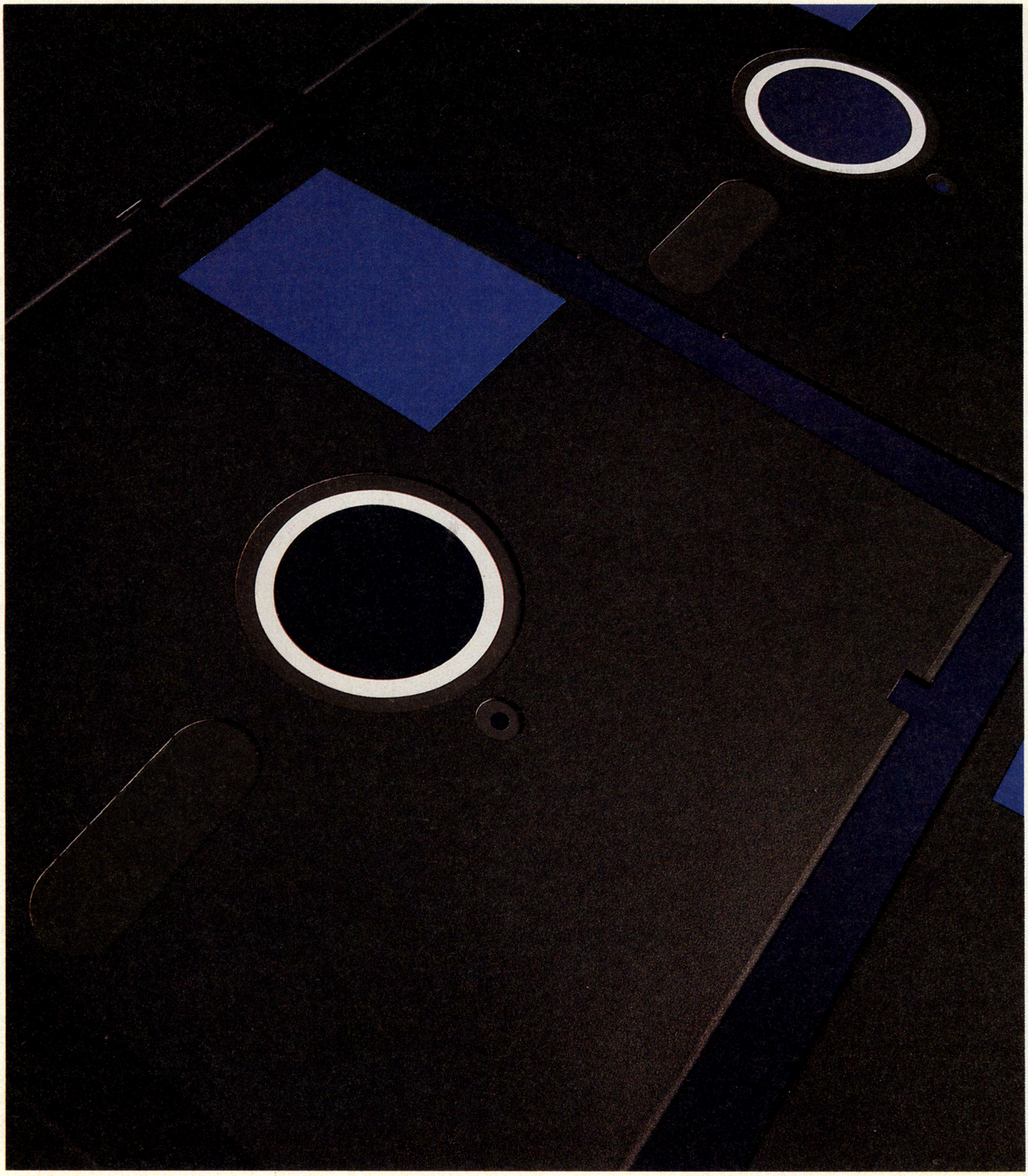
Clear the screen by typing **cls**. Type and enter the first two command statements exactly as they appear on the screen shown at right. You should see a wedge shape (like a slice of pie) outlined in a contrasting color on a light-blue screen. Now type and enter the last statement. A complete pie should appear around the slice. If you do not get these results, check for typing errors, clear the screen and try again.



PAINT

Keep the pie and wedge shapes from the last exercise *(above)* on your display monitor. Type the last command statement as it appears at right and press **Enter**. The wedge-shaped portion of the pie should turn pink. If it does not, check your typing for errors, clear the screen and re-create the shapes from the screen above by following those commands. Then re-enter the command statements at right.





The Operating System: Directions from a Disk

Your PCjr is a versatile machine, capable of performing simple tasks such as arithmetic or complicated ones such as arcade-style games. Each new task has its own set of instructions. These instructions — called software — tell the computer exactly how to use its circuits and mechanisms — the hardware — in order to do the job.

An individual piece of software, called a program, directs the computer at each step of the job. BASIC, the language built into the computer's read-only memory, is one type of program. Most programs for the IBM PCjr are stored on separate disks or cartridges, to be loaded into the machine when required. Some program disks are packaged with the computer; others you will buy later, as you need them. You can also fill disks with programs that you create yourself.

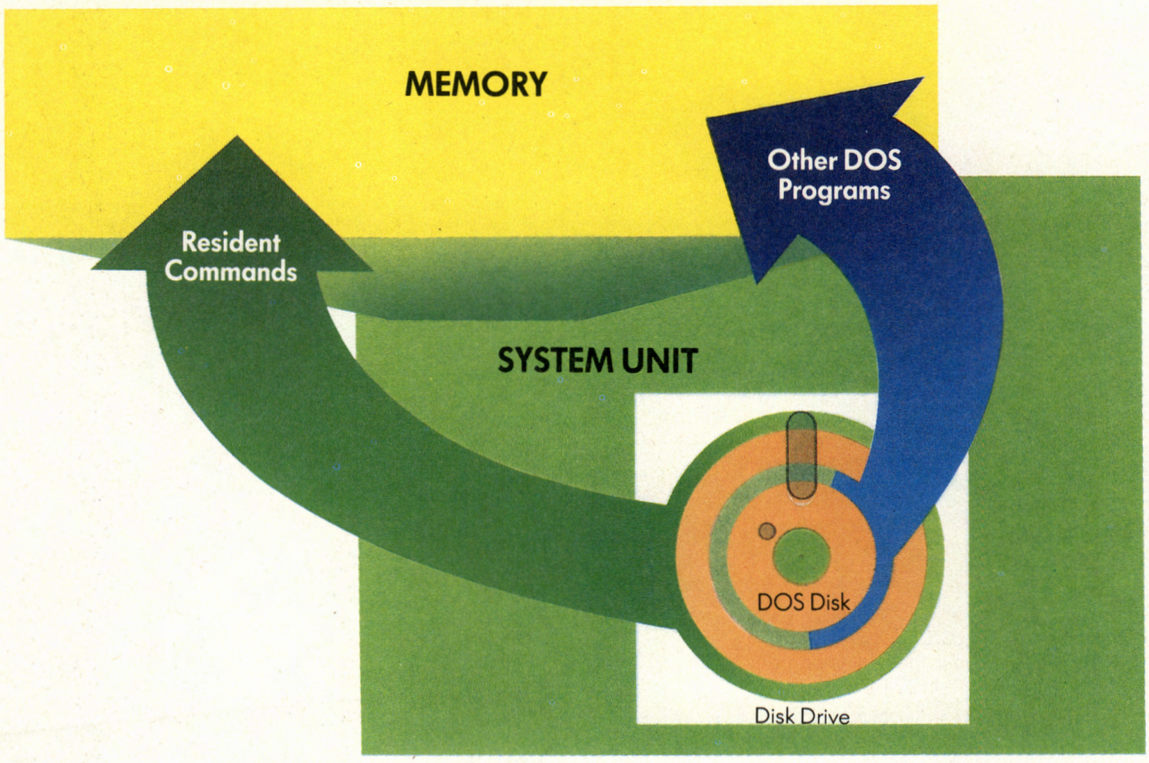
Several levels of software are used with any computer, beginning with the set of programs called the operating system. These programs give the computer fundamental instructions: how to interpret information received through the keyboard or other input devices, how to process and store information, and how to display information on the monitor or other output devices.

Higher-level software called application programs *(Chapters 4 and 5)* instruct the computer to perform complicated tasks such as word processing. Application programs tell the computer how to manipulate information to yield the desired results. But they still depend on the programs in the operating system to do the basic work of taking information in, storing it and putting it out. You must first load the PCjr's operating system into read-and-write memory every time you want to use an application program, unless the program is self-loading *(page 47)*. In some cases you must also insert Cartridge BASIC.

The operating system for the IBM PCjr is called *PC-DOS;* these initials are IBM's proprietary name for Personal Computer Disk Operating System. Several versions of DOS have been introduced, as IBM has updated the operating system and tailored it to new models of the personal computer. The version of DOS that IBM supplies with your PCjr is identified on the disk and in the manual for its use. To explore in detail the use of DOS for sorting and storing information and running the system, read the IBM *DOS* manual. As you learn more about DOS, you will find the manual an invaluable tool.

Though identical in appearance, these floppy disks for the PCjr hold completely different contents. The disks store sets of instructions — called software — that tell the computer how to do its many jobs.

Programs to Control the Computer



The Disk Operating System (DOS) is a set of programs, stored in files on a disk provided by IBM. Certain command and control programs, known as resident commands, manage information processing and the computer's links with input and output devices. The computer reads resident commands from the disk into memory, where they remain as long as the computer is on. Other DOS programs are brought into the computer's memory only when you enter a command that requires such a program; when no longer needed, these transient programs are erased from memory but remain available on the disk.

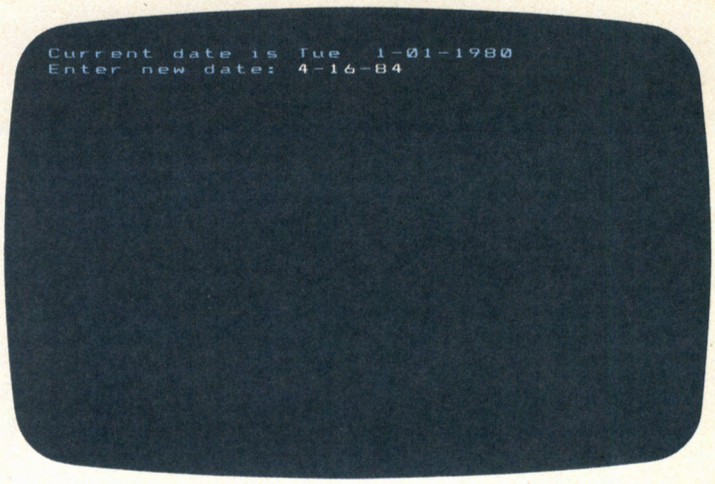
The Disk Operating System (DOS) gives the IBM PCjr instructions for basic machine tasks, such as loading an application program into memory, reading input from the keyboard and writing information on a disk. When you want one of these jobs (called housekeeping functions) done, you give a simple command to DOS, and DOS tells the computer what to do, step by step.

DOS performs this intermediary role for application software, such as an educational or word processing program, eliminating the need for the program's manufacturer to include housekeeping instructions on the program disk. For convenience, you can copy DOS programs onto an application software disk, where they will be available when needed.

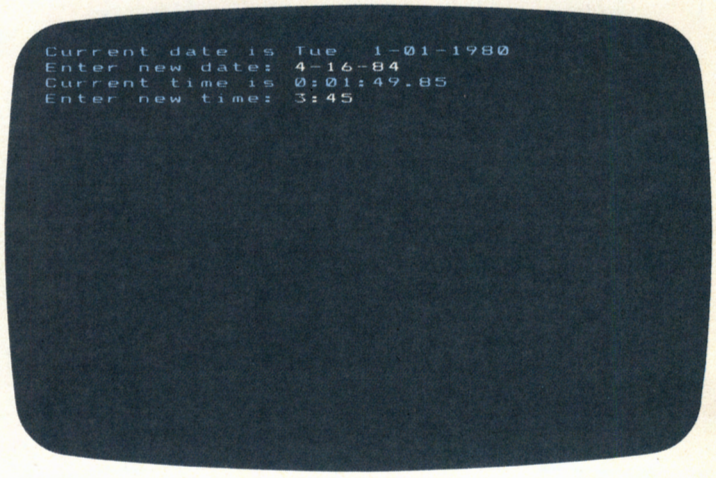
You put DOS programs in control of the computer by a process that is called booting (because DOS seems to pull itself into the computer by its own bootstraps). Instructions built into the computer tell it to look in its disk drive for further instructions. If the computer finds a disk, it then looks for a special DOS program called the boot record and reads the program into memory. The boot record then tells the computer where on the DOS disk to find the other programs needed to control the system. The computer reads these so-called resident commands *(illustration above)* into memory, where they stay until you switch off the computer.

Date and Time: Starting up DOS

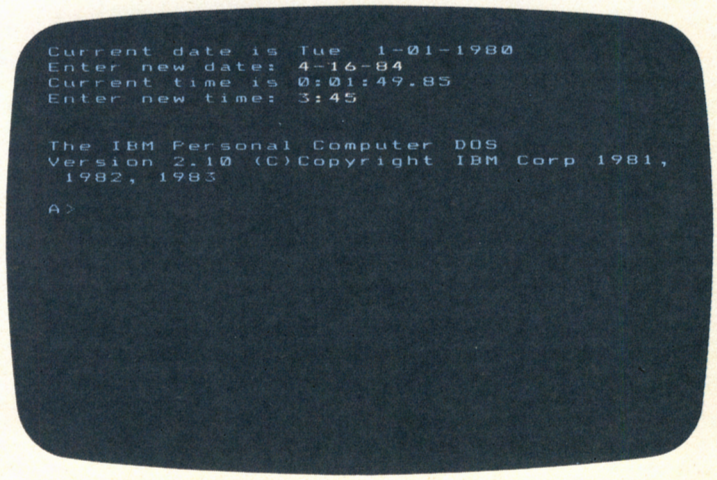
Insert the DOS disk in the disk drive (if you have a second disk, labeled *Supplemental Programs,* ignore it for now). If the computer is off, turn it on; if it is already on, reset the system by pressing the **Ctrl**, **Alt** and **Del** keys simultaneously. When the computer has brought the DOS programs into memory, a prompt will appear on the monitor. Enter the date as shown in white *(right),* using either slashes (/) or hyphens (-). Do not enter the day of the week. If the computer responds "Invalid date," try again.



The time prompt shown at right indicates that the computer has accepted your date entry. Enter the time, using the form shown in white. You may enter just the hours and minutes, although the computer can accept a time entry accurate to hundredths of a second, as indicated in the prompt. Enter the hour according to the 24-hour clock, adding 12 to the hour if the time is after noon.



The *A>* prompt *(right)* indicates that DOS is ready to carry out your commands and will refer to the disk drive for all the program files it needs, until you tell it to do otherwise. The primary floppy disk drive in a system is conventionally designated the A drive; a second drive is labeled the B drive. In single-drive systems, the terms *A drive* and *B drive* can refer not to separate drives but to different disks you insert in or remove from the one drive *(pages 38-39).*



A Disk's Table of Contents

dir a:/p ↵ causes DOS to display a directory — similar to the one at right — of the contents of the disk in the A drive. The command can be in upper- or lower-case letters, but be sure to insert spaces, colons and slashes exactly as shown. Each entry identifies one file, the amount of space (in bytes) it occupies, and the date and time it was created. Adding **/p** (for pause) to the **dir** command stops the listing of files when the screen is filled; press any character key to make the display turn to the next page.

```
A>dir a:/p

COMMAND   COM     17792   10-20-83   12:00p
ANSI      SYS      1664   10-20-83   12:00p
FORMAT    COM      6912   10-20-83   12:00p
CHKDSK    COM      6400   10-20-83   12:00p
SYS       COM      1680   10-20-83   12:00p
DISKCOPY  COM      2576   10-20-83   12:00p
DISKCOMP  COM      2188   10-20-83   12:00p
COMP      COM      2534   10-20-83   12:00p
EDLIN     COM      4608   10-20-83   12:00p
MODE      COM      3139   10-20-83   12:00p
FDISK     COM      6369   10-20-83   12:00p
BACKUP    COM      3687   10-20-83   12:00p
RESTORE   COM      4003   10-20-83   12:00p
PRINT     COM      4608   10-20-83   12:00p
RECOVER   COM      2304   10-20-83   12:00p
ASSIGN    COM       896   10-20-83   12:00p
TREE      COM      1513   10-20-83   12:00p
GRAPHICS  COM       789   10-20-83   12:00p
SORT      EXE      1408   10-20-83   12:00p
FIND      EXE      5888   10-20-83   12:00p
Strike a key when ready . . .
```

dir a:/w ↵ causes DOS to display the directory in a wide format of five columns across the screen. Only the titles of the files appear, allowing a quick survey of the contents of the disk. You may also use **dir/w** ↵ (or, in the case above, **dir/p**); if you don't use a drive specifier such as **a:**, the computer automatically searches its default drive, the drive in which it is operating.

```
A>dir a:/w

 Volume in drive A has no label
 Directory of  A:\

COMMAND   COM    ANSI     SYS      FORMAT
 COM     CHKDSK   COM     SYS       COM
DISKCOPY  COM   DISKCOMP  COM      COMP
 COM     EDLIN    COM     MODE      COM
FDISK     COM   BACKUP     COM     RESTORE
 COM     PRINT    COM     RECOVER   COM
ASSIGN    COM   TREE       COM     GRAPHICS
 COM     SORT     EXE     FIND      EXE
MORE      COM   BASIC      COM     BASICA
 COM
        23 File(s)    28672 bytes free

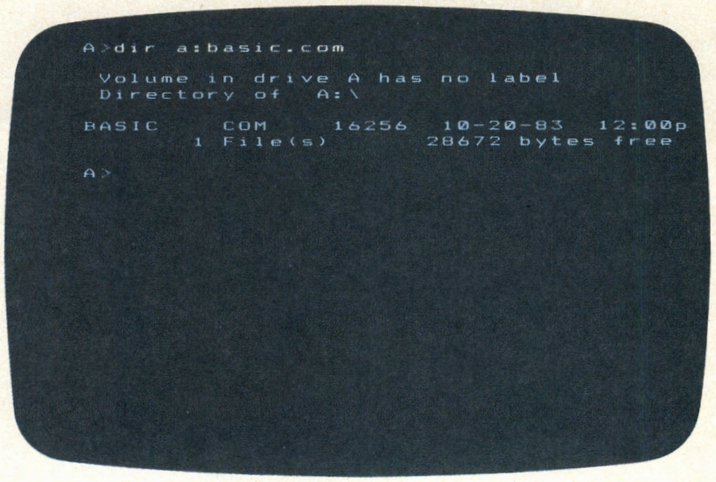
A>
```

DOS organizes and stores related information in files on disks, in much the same way that paper files are stored in a cabinet. But the magnetic coding that stores information on a disk is invisible; you can't find out what's in the files by rummaging through them. Instead, you can ask DOS to tell you the contents by using the **dir** command to display on the monitor a list — or directory — of the files stored on the disk.

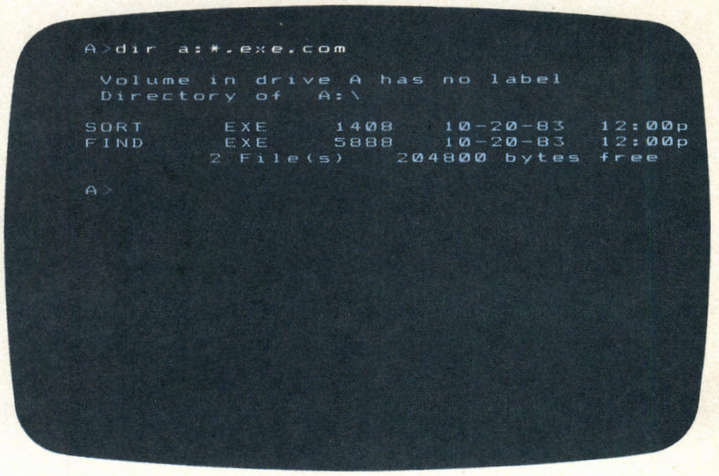
The **dir** command comes in handy when you need to find a file on a disk that is crowded with information. A single-sided floppy disk can hold up to 64 files and a double-sided disk holds 112 files — far too many to remember or even to write down conveniently. You can instruct DOS to do the searching for you, using the variations of the **dir** command shown above to display the directory in an abbreviated form, to search for an individual file or to list a group of related files. The version of DOS labeled DOS 2.10 also has a provision for subdirectories, so you can group related files under a single title in the main directory.

Files on the DOS disk and other program disks were created, named and recorded by the software manufacturers. You will have to do the same when you compile your own information on blank disks: It must all be stored in named files. Once you provide the names *(box, right)*, DOS will take care of organizing the files.

dir a:basic.com ↵ causes DOS to display the directory entry for the file you request (in this case *basic.com*), so you can be sure it is on the disk and find out its size and time of entry. Any individual file can be called up in this manner.



dir a:*.exe ↵ causes DOS to display all the file titles that have a specified part in common (in this case *.exe*). An asterisk is a "wild card," representing any combination of characters appearing in the name or the extension of a filespec. Typing the command **dir a:account.*** ↵, for example, would list all the file titles that begin with *account*. If either part of the name is omitted in the **dir** command, the computer reads the character * and will list the files accordingly.

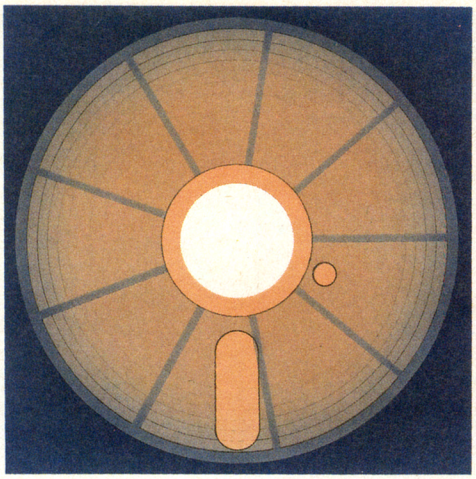


How to Name Your Files

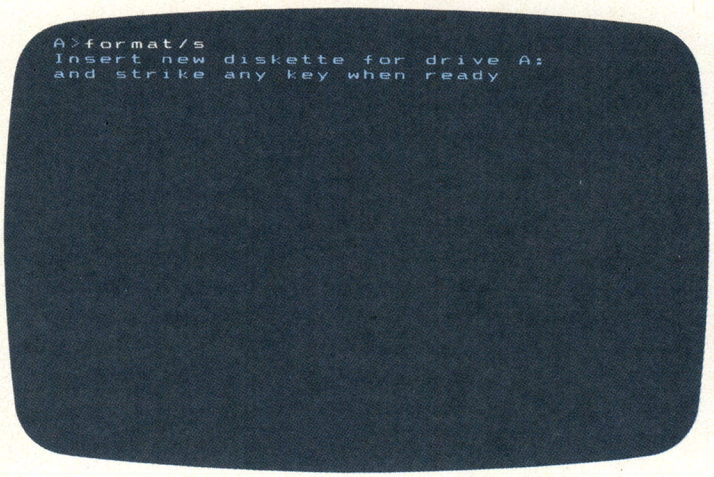
DOS recognizes file titles (IBM calls them filespecs) only if they are written in a particular way, with special attention to spaces and punctuation. A filespec must include a filename of one to eight characters; this may be followed by a period and an extension of one to three characters. Any mix of letters and numbers is acceptable to DOS (but avoid symbols such as @, \ and # in filespecs; some have special meanings to DOS, and it may respond to commands you didn't mean to give).

DOS can remember *xxplm%&* as easily as *rufdraft*, but you should choose a name that reminds you of a file's contents. You can use an extension (or filetype) as a guide to the content of a file; for example, you might use the extension *.cor* for all correspondence. Extensions are also helpful when you want to find related files. Be sure to include the extension when you call a file up; for most DOS commands, the extension — or the substitute character * *(screen, above)* — is a necessary bit of identification.

Magnetic Patterns for Easy Filing



A FORMAT FOR FILES
DOS organizes information on a disk by storing it in a pattern of arc-shaped areas defined by magnetic coding. The **format** command records this pattern on a blank disk, creating 40 concentric tracks and nine wedge-shaped sectors. Before storing files on a disk, you must format it, using DOS.

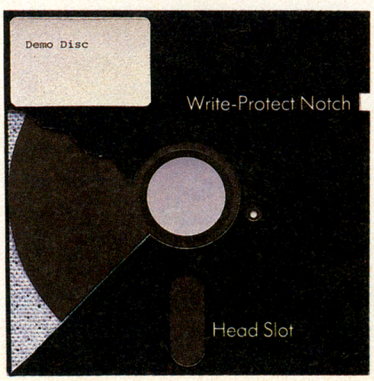


HOW TO FORMAT WITH ONE DRIVE
format /s↵ begins the formatting sequence in a one-drive system like the PCjr's, with the DOS disk in the drive. DOS reads the format program from the disk, stores it temporarily in memory, then prompts you to insert a new disk in the drive. Remove the DOS disk, insert a blank disk and press any character key. Adding **/s** causes DOS to transfer command and control files to the new disk, which then will be self-loading; you can load it into the computer without using the main DOS disk.

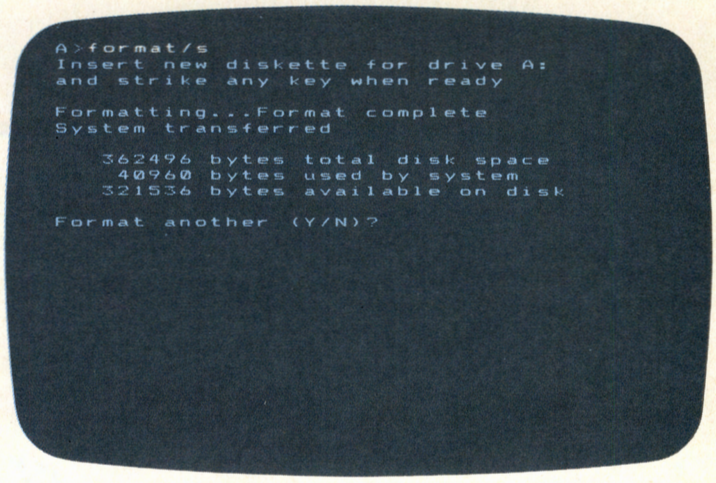
Handling Floppy Disks

The PCjr can store information on flexible plastic disks, known as floppy disks to distinguish them from the rigid "fixed disks" used in some computers. Handle a disk only by its jacket, and always store it in its paper sleeve. Keep disks away from heat sources and equipment that generates magnetism, such as telephones and tape recorders.

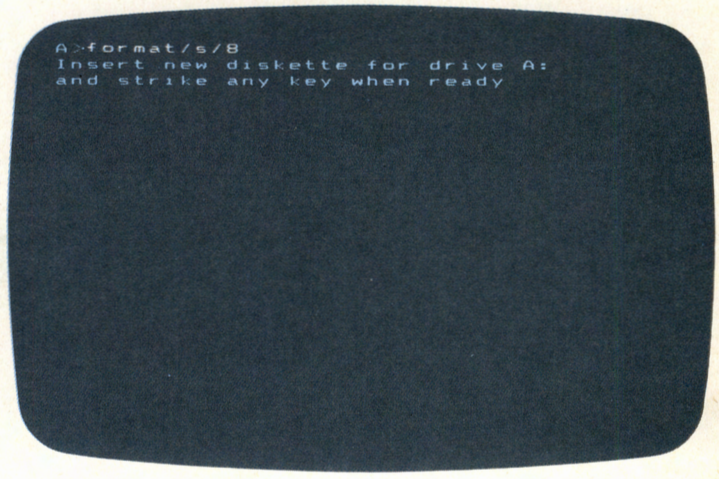
To prevent accidental erasure, cover the write-protect notch *(right)* with a gummed tab provided by the maker. When this notch is covered, the disk drive can read information from the disk through an opening called the head slot, but it cannot record or erase anything.



While the format is being recorded, the message "Formatting . . ." will appear on the monitor. If memory cannot hold all the command and control files at once, the machine will tell you to reinsert the DOS disk until all the files are transferred. When this is done, "System transferred" appears on the monitor, plus a note about disk space filled by the DOS files and a query about repeating the sequence. If you want to format another disk, type **y** and proceed; if not, type **n** and return to normal DOS control.



FORMATTING FOR EIGHT SECTORS
DOS 2.10 will create nine-sector patterns unless you give other instructions. Adding **/8** to the **format** command *(screen, right)* will make the target disk compatible with earlier versions of DOS that read only an eight-sector pattern. By adding **/1** you can also tell DOS to format a single-sided disk usable in single- or dual-sided drives.



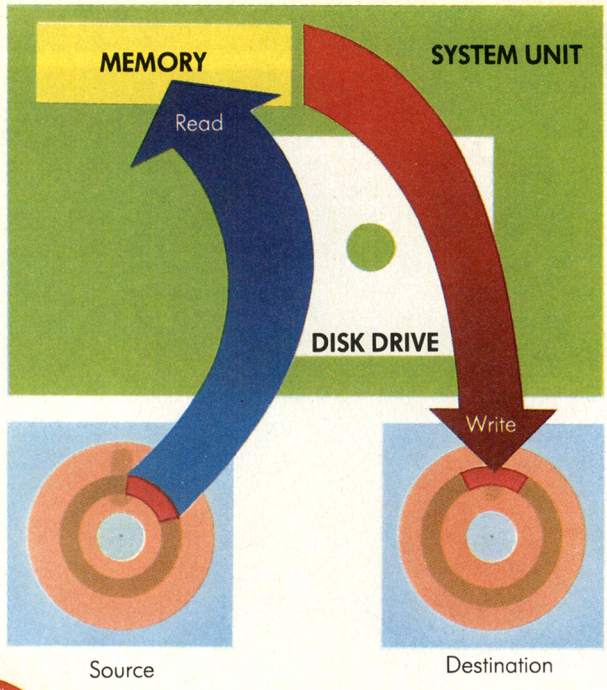
Some floppy disks hold more than 360,000 characters, the equivalent of about 125 typed pages. Sifting through that much information for a file could cause delays; to avoid them, DOS uses a reference system — a format — on each disk. The **format** command causes DOS to record on a disk a pattern of magnetic tracks and sectors *(diagram, opposite, top)*. DOS will then store information, note the track and sector of its location, and be able to find it again. The **format** command also causes DOS to find out whether the disk and the target drive are single- or dual-sided (see "FORMAT Command" in your IBM *Disk Operating System* manual), and to format the disk accordingly. Disks formatted for single-sided use will work in the PCjr's drive, which is dual-sided. But when a disk is formatted for dual-sided use, space is allocated differently than for a single-sided drive; therefore you cannot use a dual-sided disk in a single-sided drive.

The **format /s** command illustrated here makes DOS do double duty, causing it to transfer command and control files to the new disk as well as formatting it.

WARNING: Formatting a disk erases all information stored there. Usually you need to format a disk only once — before using it the first time. If you reformat an old disk, be sure that it holds no valuable files.

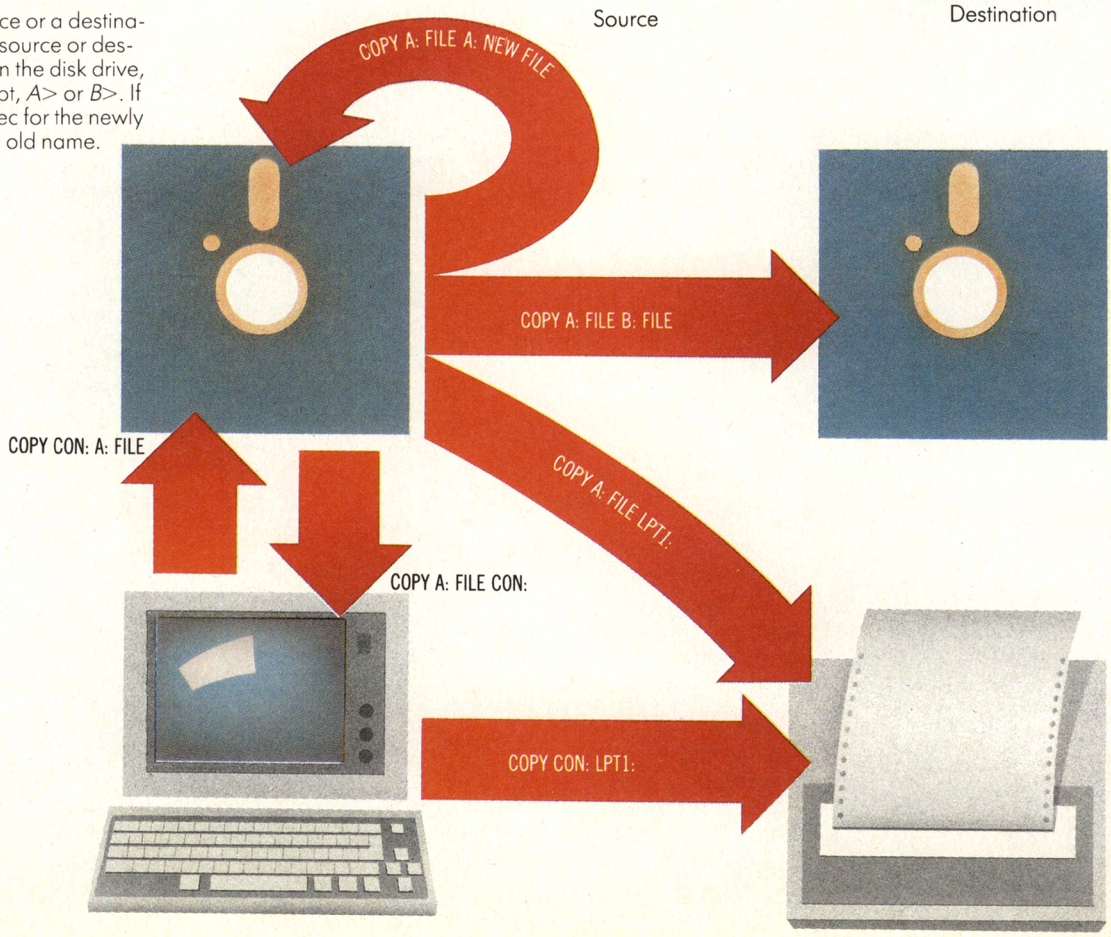
Copying Files to Transfer Data

The **copy** command, illustrated in the diagram at right, causes DOS to read information from a source, store it in the computer's memory and then write a copy at the destination. **Copy** is commonly used to copy files from disk to disk, but you can use it with a variety of sources and destinations.

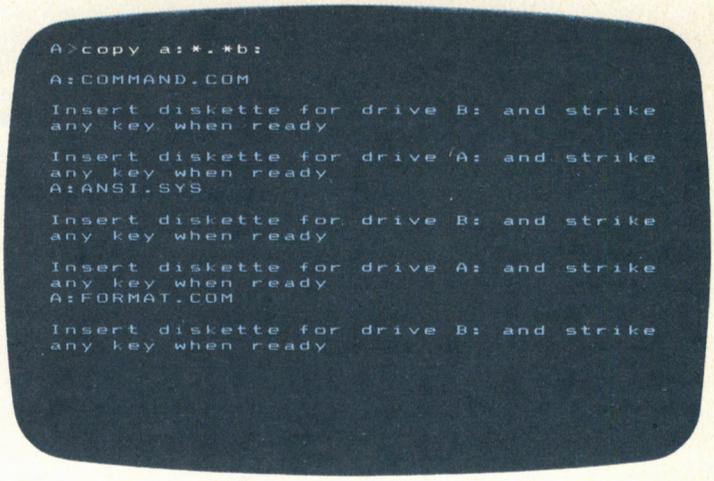


You can use the **copy** command to transfer information between system devices, as shown at right, by indicating the source and destination. Each device is known to DOS by a device name; note that a device name always ends with a colon, to differentiate it from a file title.

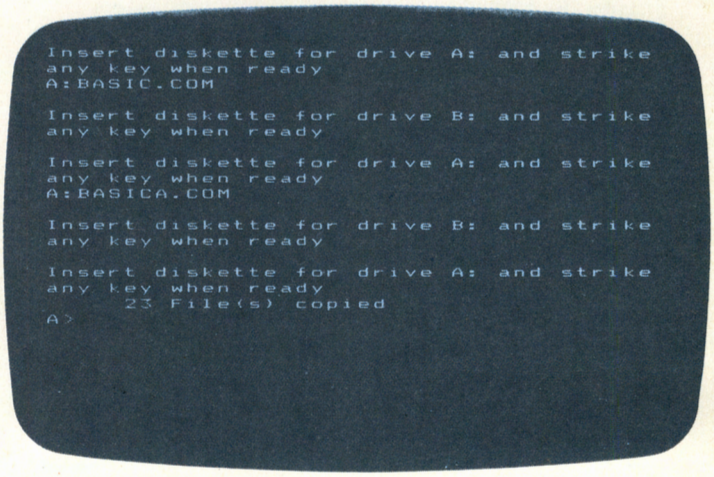
If you do not name a source or a destination, DOS assumes that the source or destination is the disk currently in the disk drive, indicated by the DOS prompt, *A*> or *B*>. If you don't enter a new filespec for the newly copied file, DOS will use the old name.



To make a backup copy of the DOS disk, put the DOS disk into the A drive and type **copy a:*.*b:** ↵ to begin the copying process. The expression ***.*** — with the wild-card asterisk in both parts of a filespec — means "all files." The codes A: and B: here refer to source and destination disks. DOS reads the file from the source disk (A:), stores it in memory and then tells you to insert a disk "for Drive B:". Remove the source disk and replace it with the destination disk (B:) and press any character key.



As the computer copies files from one disk to the other, it displays a list similar to the one at right. You will have to switch the source and target disks 23 times, once for each DOS file. When the copying is complete, the computer reports how many files it copied.



Your first use of the **copy** command will be to make a backup copy of the DOS disk, so that you won't be left without an operating system if something happens to the original. The versatile **copy** command will also become one of your most useful tools for moving and storing all your files, and for printing and displaying files received from keyboard input or from other computers *(pages 67-73)*.

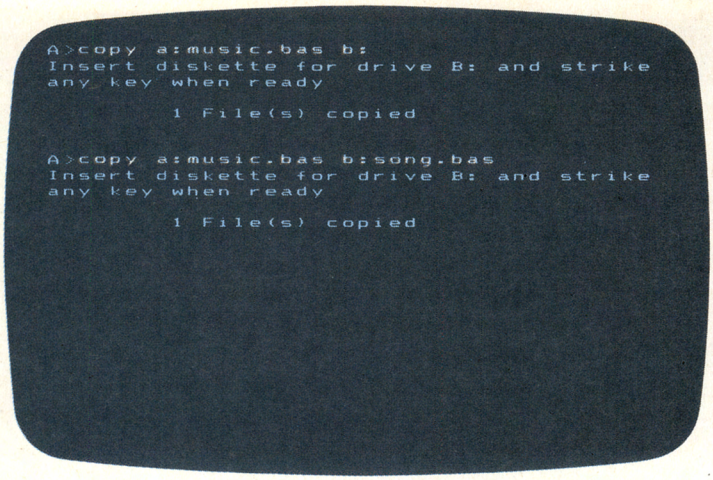
Because the **copy** command has so many uses, you must compose it carefully. To give you the results you want, DOS must know the intended source, the intended destination and the filespec of the information you want to copy. The source and destination are entered as device names: acronyms identifying input and output devices *(opposite)*.

A properly composed **copy** command always starts with **copy** followed by the name of the source device. Next comes the filespec of the source file, then the name of the destination device and finally the filespec by which the copied file will be known. Remember to differentiate between the letters *l* and *O* and the numerals *1* and *0*.

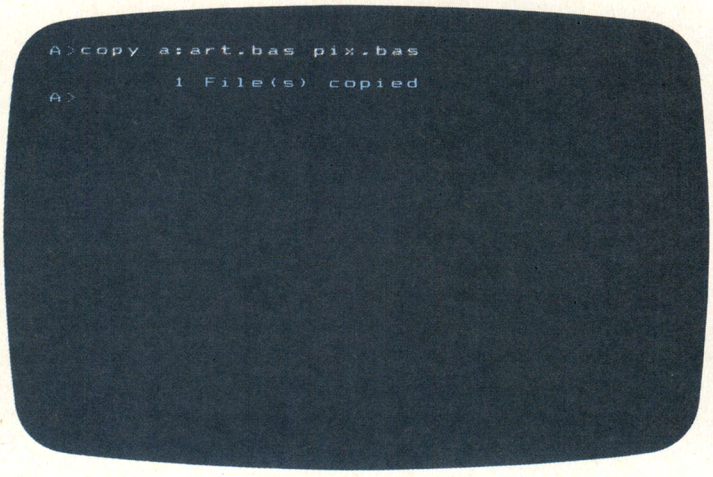
Pay attention to filespecs when you are copying. Each file on a disk must have a unique filespec; if you try to store new information under a filespec already in use, DOS will erase everything it has previously stored under that filespec as it writes the new information.

A Variety of Uses for the Copy Command

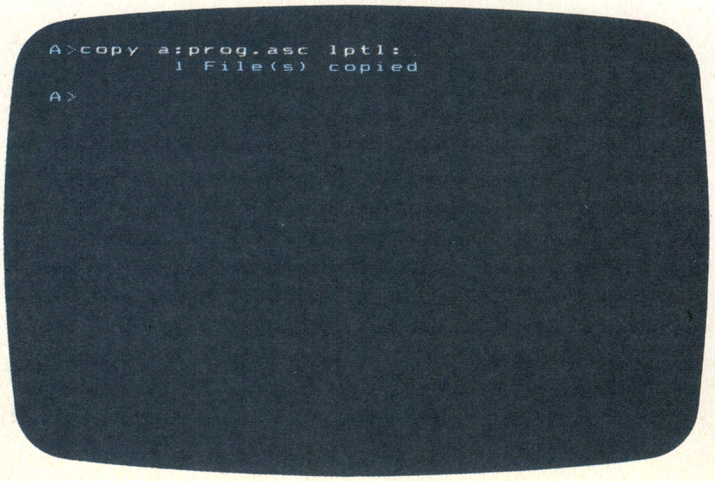
copy a:music.bas b: causes DOS to copy a file (in this example, *music.bas*) from one disk to another, where it will be stored with the same name. Entering the command **copy a:music.bas b:song.bas** would result in the copied file's being stored on the target disk with a new name — *song.bas*.



copy a:art.bas pix.bas ↵ causes DOS to copy a file (in this case, *art.bas*) to the same disk, but with a different name (here, *pix.bas*). The result is two files on the disk that are identical in all but name — useful if you want to make changes to a file while retaining a copy of the original.



copy a:prog.asc lpt1: ↵ causes DOS to copy a disk file — in this case, *prog.asc* — from the disk in the drive to the printer. This is useful when you want to have a paper record of a disk file that has been stored in a readable format — such as a file created by copying keyboard input *(opposite)* or a file of text sent from another computer *(page 70)*.



Copying Files from the Keyboard

copy con: a:frog.ltr ↵ causes DOS to save information entered at the keyboard (or console) to store in a file on a disk in the **a:** drive — in this case, under the name *frog.ltr*. When you enter this command, DOS begins saving all keyboard input in memory, for later transfer to the disk file. This is useful for saving a short file, such as a memo, or a list of commands for DOS to execute later.

```
A>copy con: a:frog.ltr
Calaveras County, June 22

Dear Luther,
```

As you type, you must occasionally end a line by pressing the **Enter** key (↵). If you fail to do so, the computer will make a buzzing sound and may appear to stop operating; it cannot process extremely long lines. Even though it seems to start new lines by itself, the computer reads what you type as a single line until you type ↵.

Your signal to DOS that the entire file is ended is to type ^**Z** ↵ (press **Control** and **Z** simultaneously; release them, then press the **Enter** key). When it receives this end-of-file character, DOS transfers the file from memory to the named disk file.

```
A>copy con: a:frog.ltr
Calaveras County, June 22

Dear Luther,

This is just a reminder that you should
pick up the posters on the Monday before
the race.  Please bring them directly to
the office.

Regards, Hester
^Z
        1 File(s) copied
A>
```

copy con: lpt1: causes DOS to copy information entered at the keyboard directly to the printer. As in the case of a file copied from the keyboard to a disk, DOS saves the information in memory until you type the end-of-file character, ^**Z** ↵. Then the entire file is sent to the printer. This command is useful when you want a paper copy of the information but don't need to save it on a disk, as in the case of a short memo.

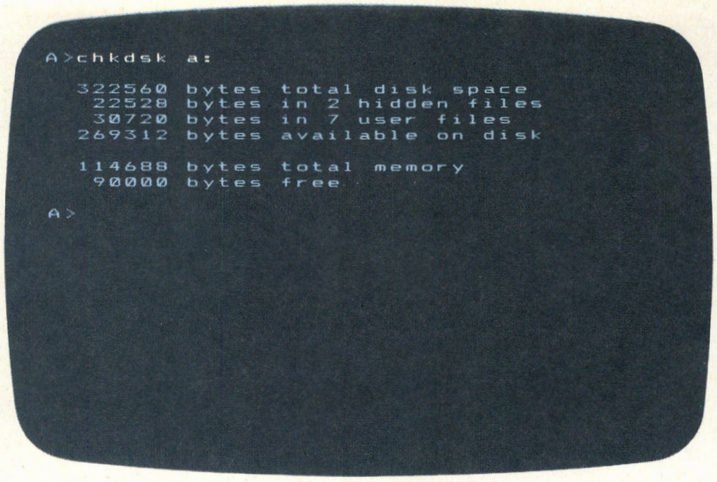
```
A>copy con: lpt1:
Dear Luther,

Thanks for assisting with the posters.
I am sure they made a great contri-
bution to the turnout for the great
race.  I hope you can work with us
next year.

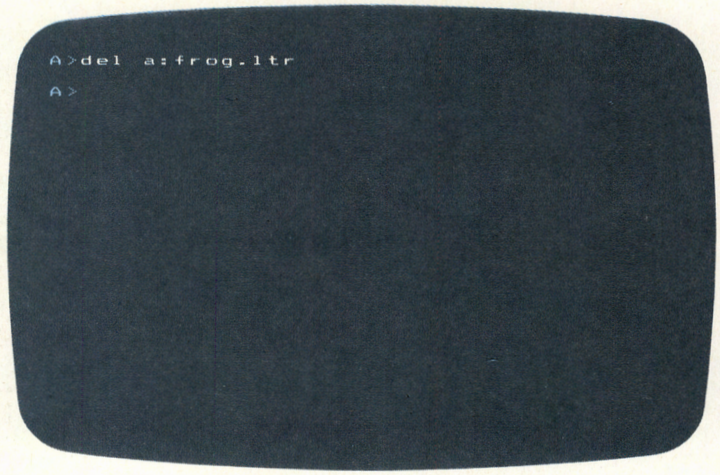
Regards, Hester
^Z
        1 File(s) copied
A>
```

DOS Commands for Other Tasks

chkdsk a: ↵ causes DOS to scan the contents of the designated disk and report the number of files on it and the amount of space left. It is important to keep track of the free space on each disk, since some programs may lose data when there is not enough space available to save a file.



del a:frog.ltr ↵ causes DOS to delete the designated file, in this case *frog.ltr*. Typing the command **erase a:frog.ltr** ↵ has the same result. These commands are useful for eliminating outdated files, making room on the disk for more data.



As you gain more practice with DOS, you will discover more commands that help you create and handle files. Some of these are shown above and at right; many others are explained in the IBM *DOS* manual.

One command not shown here is **diskcopy.** This command is similar to **copy *.*** *(page 39)*, but it does more than simply copy the files. **Diskcopy** erases everything on the target disk and then transfers the format *(page 36)* and the operating system in addition to the files.

Another command you may need is **mode**, which can set up the computer to use a nonstandard printer, or can change the way the monitor displays information, as required for some graphics uses.

WARNING: If you have disks that have been formatted in different versions of DOS, do not mix programs from different versions on one disk. If you do so, and then use *chkdsk.com* or *diskcopy.com*, incompatible programs can erase data from the disk you are checking or copying. If a disk drive spins continuously during *chkdsk* or *diskcopy*, incompatible programs may be the explanation. Switch off the computer, remove the disks, and then use the *copy* command to put the compatible *command.com*, *chkdsk.com* and *diskcopy.com* programs onto the affected disk.

rename a:frog.ltr green.ltr ↵ causes DOS to change the name of a file, in this case from *frog.ltr* to *green.ltr*. If you type **ren frog.ltr green.ltr** ↵ you will get the same result. These commands affect only the names of files; they have no effect on the data stored in the files.

```
A>rename frog.ltr green.ltr

A>
```

type a:green.ltr ↵ causes DOS to display the contents of the designated file on the monitor. This command lets you find out quickly what is in a file. If the file was created using an application program, the **type** command may result in a display that contains seemingly meaningless graphics symbols. These symbols represent special codes that some programs insert in files for their own housekeeping purposes.

```
A>type green.ltr
Calaveras County, June 22

Dear Luther,

This is just a reminder that you should
pick uo the posters on the Monday before
the race.  Please bring them directly to
the office.

Regards, Hester

A>
```

sys b: ↵ causes DOS to copy the operating system files from the DOS disk to another disk. You will use this command to make an applications program disk self-loading — to prepare it so that you can use it without first loading DOS into memory. The prompt, "Insert a diskette for drive B:" means that you should remove the DOS disk and insert the applications disk. Most applications programs also require that you add the DOS *command.com* file; after the **sys** command is executed, type **copy a:command.com b:** to complete the preparation of the applications program disk.

```
A>sys b:
Insert diskette for drive B: and strike
any key when ready

Insert diskette for drive A: and strike
any key when ready

Insert diskette for drive B: and strike
any key when ready

System transferred
A>copy a:command.com b:

Insert diskette for drive A: and strike
any key when ready

Insert diskette for drive B: and strike
any key when ready

        1 File(s) copied
```



Education and Entertainment

Games and learning are strands of the same rope. By playing games that imitate the world, children sharpen their physical and social skills: coordination, memory, quick reflexes (both mental and physical), the ability to work with others. Prehistoric children may have tossed sticks and pebbles on the sand, or played at hunting with crude bows and arrows. Children since have learned to experience and master the world around them by playing such games as hide-and-go-seek and soccer, dollhouses and train sets, chess and parcheesi. Today's children have added computer games.

But you don't have to be a child to enjoy playing games on your PCjr, or to learn on it. Given the right software, the PCjr can challenge not only the fingers but the minds of children and adults alike.

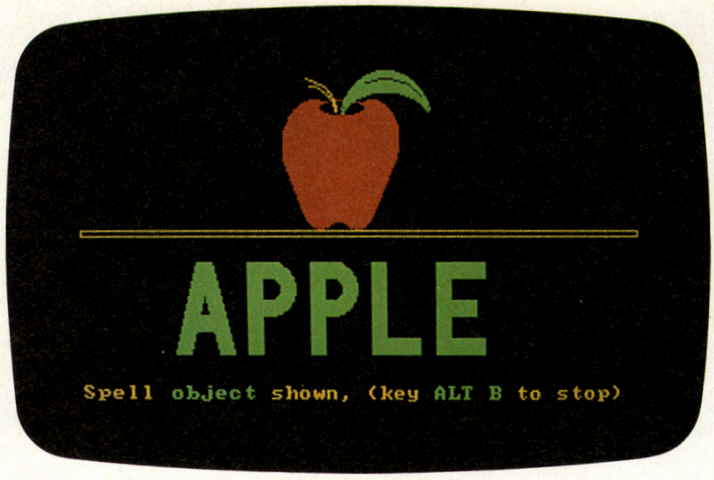
Games for the PCjr come in several guises, from the fast-action shoot-'em-ups of arcade-style entertainment, to the familiar strategies of blackjack and chess, to the structured environments of adventure simulations. Some of these adventures are text-oriented interactive fiction: You respond, via the keyboard, to a series of questions or statements. Other simulations fill the screen with vivid graphics. Perhaps more important, by having you play a role in the drama, simulations can encourage you to look at problems from a new perspective.

Simulations play an important part in educational software as well. As personal computers become more powerful and thus more versatile, such approaches as drill-and-practice and electronic text are giving way to simulations and other programs that allow more interaction between machine and student. The best educational packages not only demonstrate ideas and concepts, but also test and evaluate a learner's performance. Moreover, they present the material through colorful graphics and a strong story line.

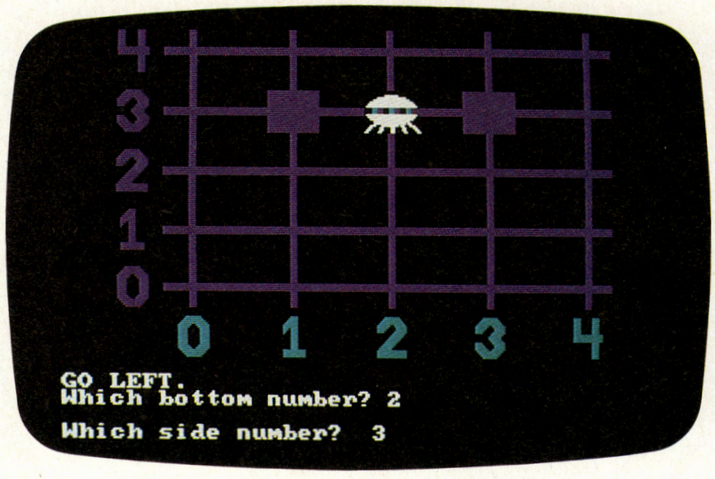
If an educational package seems too much like a glorified game, remember two things: A computer is only as good a teacher as its software allows it to be, and boring software is rarely used more than once. Consult with a schoolteacher who works with a computer for help in evaluating available programs. Whether labeled education or entertainment, the best of these software packages encourage you to learn from your mistakes — and to have fun with your computer while you do.

A high-tech version of chess can take place on a computer screen, with game pieces and a multicolored board created electronically, but three replicas of walrus-ivory chessmen from 12th Century Scandinavia serve as reminders of the game's ancient traditions. Chess programs for the PCjr can play against you at several levels of skill, or let you play against a human opponent.

Learning Games for Youngsters



This flashcard-like tutorial introduces children between two and six years old to letters, numbers and words. When a character or a picture of a familiar object or animal appears on the PCjr's screen, the child responds by typing the matching character, or by spelling the word for the number, object or animal. Very young children can ask the computer to display the words in outline *(right)* as an aid to finding the matching letters. Sound effects and an animated character celebrate correct answers.



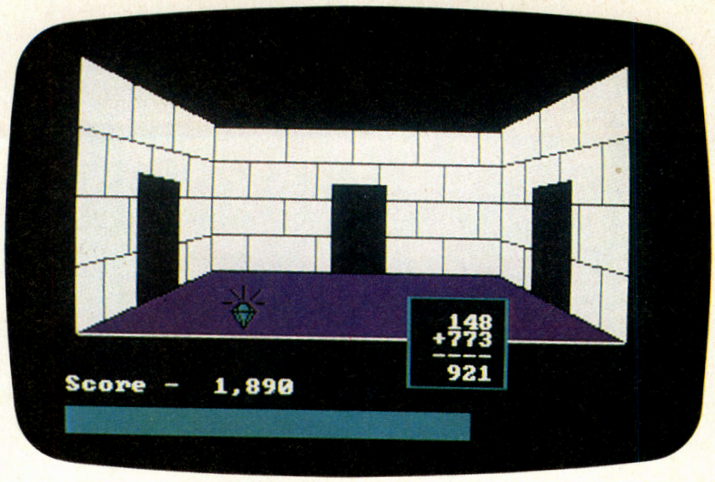
A set of six educational games, graded by difficulty, teaches four- to 10-year-olds to use pairs of numbers and letters to identify coordinates on an array or grid *(right)*. Players enter coordinates to uncover hidden numbers or cartoon characters, to play a version of tick-tack-toe, or to draw a picture. Incorrect entries prompt hints from the computer; correct answers are rewarded with colorful graphics, music and praise.

To turn your computer into a useful educational tool and a patient teacher, you must get software that is suited to your needs. As illustrated here and on the following pages, there are several kinds of programs to choose from, depending on the age of the user and the type of instruction desired.

You will want to help young learners get started, of course, but they should be able to proceed at their own pace. Look for programs that let the learner control the length of screen display time and that offer instruction for several age levels.

Check the program's method of dealing with repeated failure. Some tutorials loop the learner back to earlier instruction, some present the right answer and an explanation, and others branch the learner to an alternate treatment of the same problem. Finally, be sure the maker will replace the program disk for a nominal charge in case of damage.

An adventuresome way to improve basic mathematical skills, this game challenges learners to find their way through a castle maze of passageways and rooms *(right)*. Players may choose problems in addition, subtraction, multiplication or division. Solving the problems will open doors, win treasure and score points. The top 10 scores and players' names are listed at the top of the screen at the end of each game.



Quick math reflexes are the goal in this educational game, which pits the learner against a series of math problems, a three-headed monster and a 60-second clock. Six levels of difficulty increase the challenge of problems in adding, subtracting, multiplying and dividing. Correct answers make the monster disappear, part by part, and score points; wrong answers keep the monster alive longer.

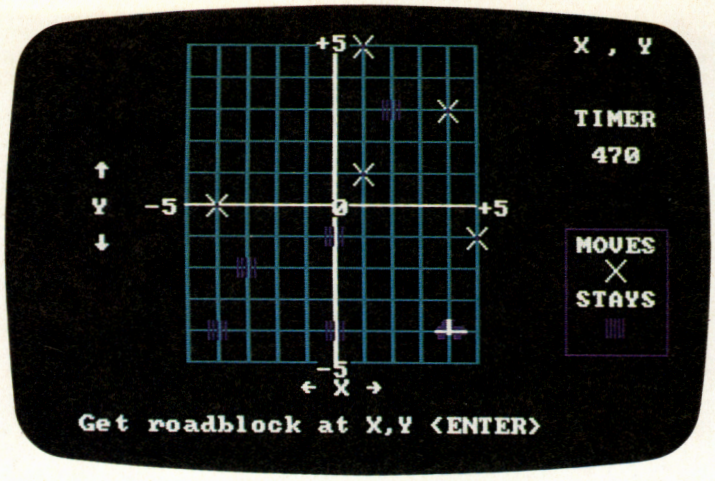


Preparing Your Program Disk

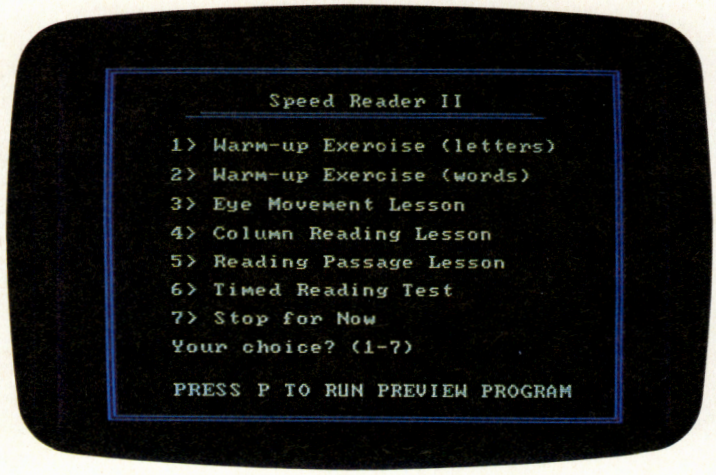
With some game and educational programs, you will have to copy the PCjr DOS command file onto the master disk to make the program self-loading; otherwise, you have to load DOS first every time you want to start the program. With other programs, the necessary boot record is already on the disk when it comes from the manufacturer. Examine the documentation carefully. It should spell out exactly what equipment you will need and describe clearly how to start the program, including step-by-step instructions for putting the DOS command file on the program disk if that step is necessary.

Make a backup copy of the master disk if possible. (Some disks cannot be copied; you get a lifetime guarantee on the master disk instead.) Use the copy and save the master; that way, if anything goes wrong, you can make another copy. Some game instructions also suggest that you format a working disk for recording your various strategies. In that case, the documentation should also explain how to save your moves for replays.

Curricula for Your Computer Classroom



Designed to promote deductive reasoning, analytical thought and creativity in eight- to 13-year-olds, this collection of five educational games in one builds on coordinate-plotting skills that have been learned in a companion game *(page 46)*. Players guess coordinates, or a range of coordinates, to trap a robber or to find hidden treasure. Mistakes elicit on-screen help statements. Correct answers win colorful graphics and musical vignettes.



Daily sessions with an interactive tutorial like the one shown at right can help improve your reading speed and comprehension. Each session includes exercises to improve perception, peripheral vision, eye span and movement, and reading speed. The program keeps track of your progress. A special editor feature lets you enter your choice of reading material, analyzes it and gives you its approximate grade level.

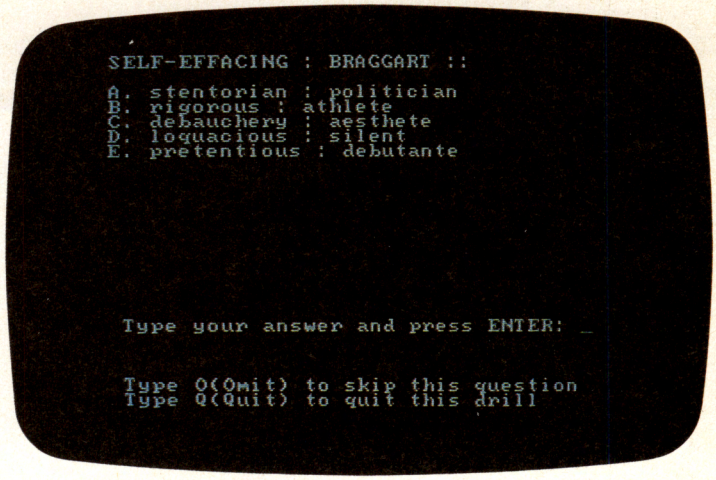
Your PCjr has many of the attributes of a good teacher: It can call upon vast amounts of knowledge, pose questions and score responses, keep track of progress and repeat teaching sequences with infinite patience. If you plan to use your PCjr for educational purposes, you should give careful consideration to two things: the software you select and the monitor you use for display.

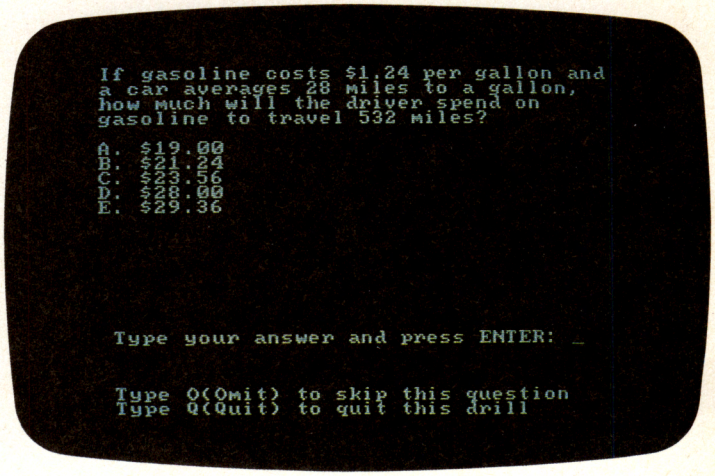
Composite-video monitors are adequate for most educational programs; they are less expensive than RGB monitors as a rule, and will generally provide sufficient resolution for most graphics. Bear in mind, however, that many composite-video monitors have trouble displaying more than 40 characters per line; if much of your material is textual, you may prefer the higher-resolution, 80-character screen of an RGB monitor.

Good educational software holds the user's interest with some of the same devices that make computer games appealing — high-quality sound and graphics, and routines that call for frequent interaction. The element of enjoyment is just as important to computer learning as it is to classroom education.

Several types of educational software are available for the PCjr. The most common leads you through practice drills, generally of information you have learned elsewhere, such as arithmetic, spelling or a language. Some drill programs keep a

Among the tutorial programs available for the PCjr are some designed to teach students how to take college-entry examinations. The examples at right are drawn from the verbal *(top)* and math *(bottom)* tutorial sections for the Scholastic Aptitude Test. Students proceed at their own pace, selecting from a lesson menu and then working in one of three modes: instruction, test or timed test. In the instruction mode, the program teaches new material by providing hints and clues if the student does not get the right answer the first time around.





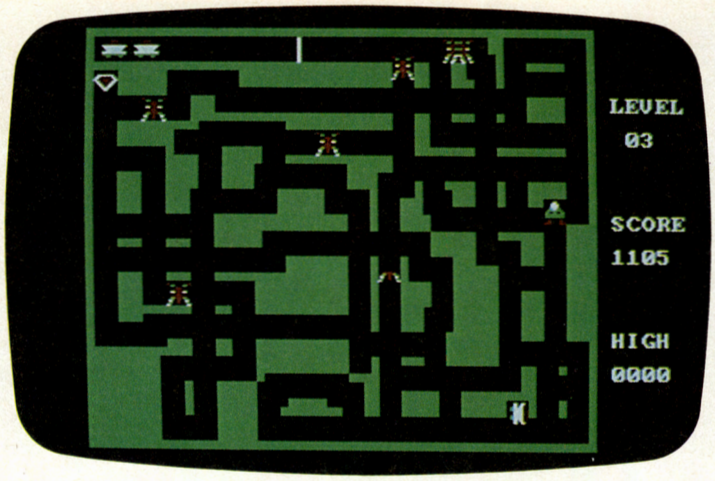
record of your correct and incorrect answers to let you keep track of your progress. Some allow you to make up your own questions and answers for later review. There are even programs that drill you on musical notes and scales and then allow you to compose your own tunes.

A second type of program, the tutorial, does more active teaching. These programs present a sequence of textual information; some periodically pose questions to make sure that you understand the concepts, and if you answer correctly, they advance and present more information. Tutorial programs are used to teach the basics of such disciplines as algebra, geography and astronomy, as well as computer languages such as Logo and BASIC.

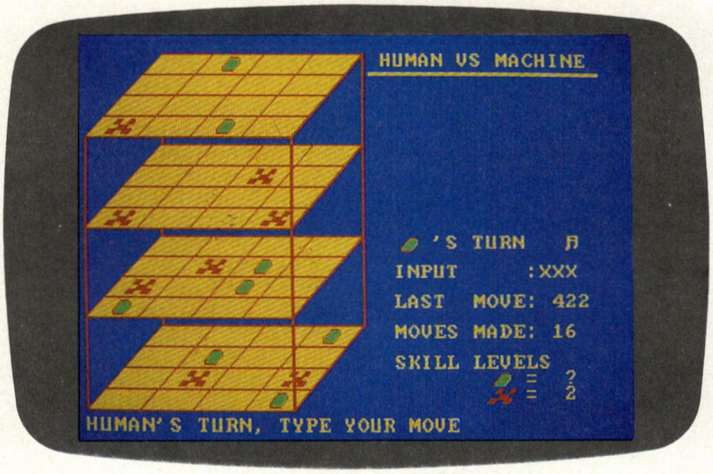
Another, more sophisticated type of program simulates real situations and their concomitant problems: You learn by working through the difficulties. This type of software might present the purchasing and pricing problems of a small business or, for children, the problems of distributing equal volumes of liquids into odd-shaped containers.

The hallmarks of good educational software, wherever you find it, are clear instructions, commands that use keys logically, and the graphics, color and sound effects that keep the learner interested.

The PCjr's Electronic Action Gameboard



To score in this fast-paced, arcade-style game, you must maneuver your mining car to gather gems, and fire your laser gun to disable the robots rampaging through the twisted shafts on each of the mine's 10 levels. Your current score and whereabouts, as well as the high score to beat, are posted on the right side of the screen. The game comes in cartridge form and may be played with the keyboard or a joystick.



With a red-green-blue (RGB) monitor, you can play this three-dimensional tick-tack-toe against the machine or another player. The object is to get some of your pieces to form a horizontal, vertical or diagonal line on any level, or through all four. The levels are numbered from top to bottom, the rows from front to back, the columns from left to right. Typing the three-digit number 234, for example, would put your piece in the second level, third row from the front, fourth column from the left.

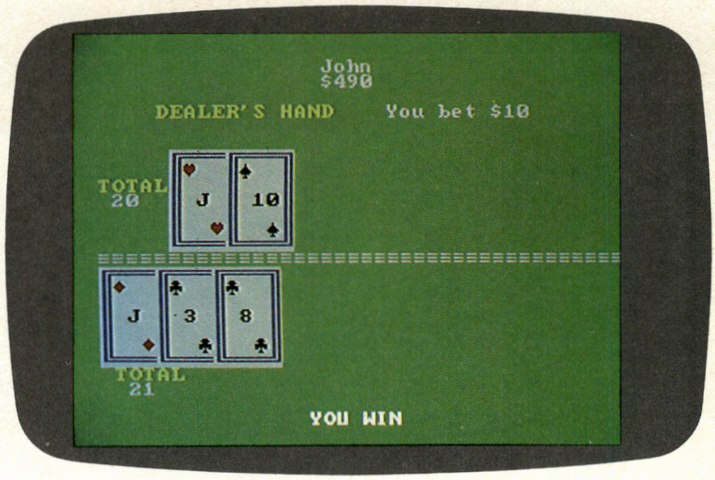
For some of the games you can play on your PCjr, joysticks are an optional controller, more flexible than keyboard controls. IBM sells a joystick for the PCjr, but you may also use one made by another manufacturer.

There are two categories of joysticks — discrete and proportional (the PCjr model is proportional). The discrete type is the simpler and less costly. It can direct the cursor or on-screen game character in only eight directions: the four points of the compass and four diagonals. It cannot control speed or distance traveled.

A proportional joystick, by contrast, can move the cursor or game character in any direction, and the speed and distance the cursor moves correspond to the movements of the joystick. With a better-quality stick, you will also be able to move the cursor or character a very small distance at a time. The majority of joysticks have a trigger or triggers mounted on the stick, for functions that vary from game to game.

Once attached to the computer, the joystick will need adjusting. For some games, you may want the stick to be self-centering — to rebound to the center position when you let go; for other games, you may want it to stay where you leave it. You may also need to calibrate the stick so its moves are coordinated with the cursor. Some games have a diagnostic screen for this; if yours doesn't, experiment to set the stick properly.

A disk drive and Cartridge BASIC let you enter the world of chance in this colorful package of three classic casino games on one disk: blackjack, slot machine and poker. Up to four players at a time may gamble, each with a $500 stake. In the card games, the computer deals; for slot machine, the player pulls the handle by pressing the keyboard space bar. The slap and snap of the cards and the whir of the slot machine accompany play. You can carry your winnings from game to game within the casino.



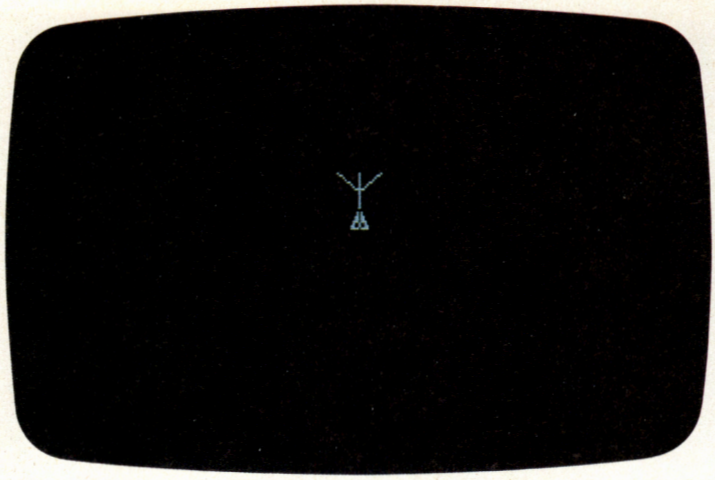
Through a series of computer-human exchanges similar to those shown at right, adventure game players pursue a goal through deserts, forests and mountains, fending off danger and trying not to get lost. As the human player learns the rules (mostly by trial and error), the exchanges become less maddening and more rewarding. Depending on the game, players may be armed with information, weapons, or words with magical properties. Solving the problems posed by these programs can take days — and sometimes weeks.



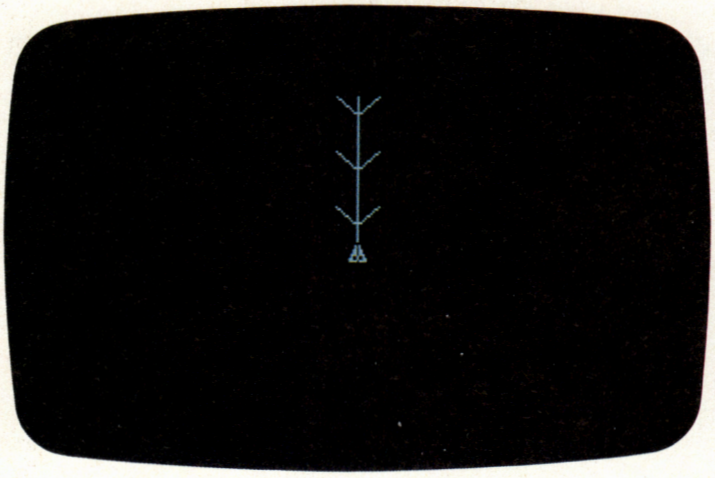
Simulations like the one at right conjure up another world and put you in charge — in this case, of a 19th Century railroad. To make your company grow, you must achieve a complex series of objectives, filling up with coal when necessary, repairing broken machinery, picking up and delivering various cargoes to keep industry and marketplace supplied. If you succeed, more track is laid — making the next series of pickups and deliveries that much more complicated.



Mastering Logo with the Help of a Turtle



Using simple commands in the computer language called Logo, you can program graphics like those shown here, starting with a small procedure — to create a twig *(right)* — and using it to build others. To define the procedure TO TWIG, you type a series of instructions, such as **FORWARD** (or **FD**) **20** or **LEFT** (or **LT**) **45**. If you then type **TWIG**, a triangular cursor called the turtle will obey each command, moving forward 20 steps or making a 45° left turn, leaving lines on the screen to draw the shape.



From the small unit of a twig, you can then build a larger unit — a branch. Again you must define the procedure — in this case, TO BRANCH. By using TWIG, the procedure already created and defined, you can define TO BRANCH in instructions that appear to be a kind of shorthand: **TWIG**, **FD 30**, **TWIG**, **FD 30**, **TWIG** — and so forth. If you then type **BRANCH**, the turtle will draw a shape like the one shown at right.

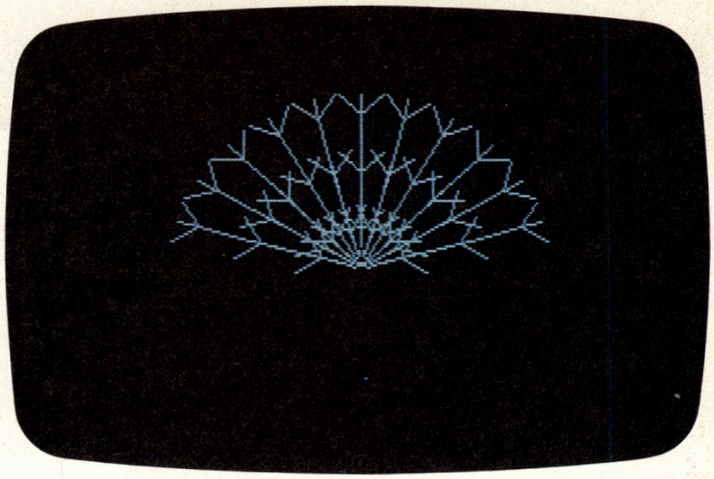
Logo is a computer language that can give you or your child a gratifying sense of mastery over the machine. In minutes you can teach the computer to do your bidding — and in the process, gain an intuitive understanding of programming.

The key to Logo's success as an educational tool is a feature called turtle graphics. The turtle — a small triangular cursor — moves in response to typed commands such as **FORWARD 10** (move forward 10 steps) and **RIGHT 90** (make a 90° right turn). As the turtle moves, it creates a design on the screen, allowing you to watch your commands in action. (Commands may be abbreviated — **FD** and **RT**, for example.)

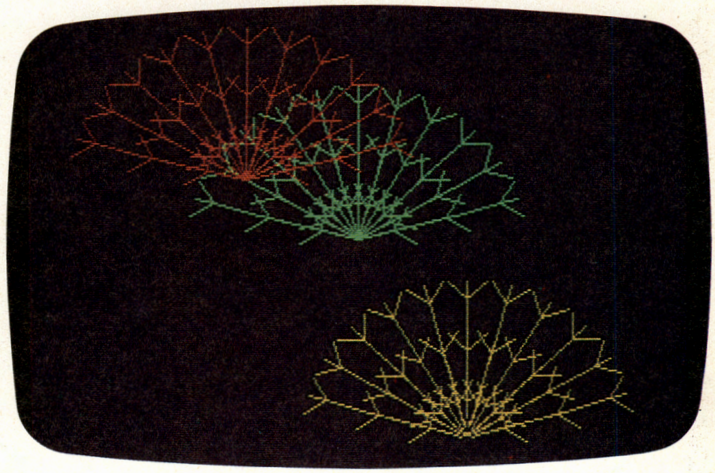
If you make a mistake — by typing **GD** or **GOWARD**, say — the turtle mildly responds, "I DON'T KNOW HOW TO GOWARD." In much the same way children learn to talk, you learn Logo's vocabulary and grammar by interacting with the turtle.

As you master simple procedures, you can use them to build more complex ones — which is the beginning of programming. First you learn to name a procedure — TO TWIG, for example, might be a procedure to draw lines that resemble a twig *(top screen, above)*. Then you define the procedure by typing each step needed to draw the shape. Once the procedure has been defined, you have only to type **TWIG**, and the turtle will draw one without further ado.

With BRANCH (which incorporates TWIG), you can now teach the turtle to draw a shrub. You define the procedure TO SHRUB by typing commands such as **LEFT** (or **LT**) **75, BRANCH, REPEAT 10 [RIGHT 15 BRANCH]**. The **REPEAT** command tells the turtle to do a procedure a given number of times. In this case, the turtle will make a 75° left turn and execute BRANCH; then it will do the two steps in brackets — turn right 15° and execute BRANCH — 10 times.



To create the last screen — a copse, or thicket of small shrubs — you use SHRUB to define the new procedure TO COPSE, just as you used BRANCH to define SHRUB. With other instructions built into Logo — such as **SETPC**, for Set Pen Color — you can tell the turtle what colors to make each shrub. If you include these instructions in the definition of the procedure, the turtle will create a colorful screen as soon as you type **COPSE**.



More Languages

Logo is one of hundreds of programming languages. Designed to teach children math, it is used for graphics programs and can also handle text. Here are brief descriptions of five other languages:

FORTRAN was the first widely used computer language. It can serve as a general-purpose programming language, but its forte is scientific and engineering applications.

COBOL was developed for business and commercial applications. It uses words and phrases similar to English, making it easy to read.

BASIC is an interactive language that gives the user an immediate response to what is typed into the computer. The most popular language for microcomputer users, BASIC is adequate for many business and commercial applications.

Pascal is useful for anyone studying computer science. It is best suited for long, complex programs.

LISP (LISt Processor), is the language of choice for artificial intelligence (AI) research because of its ability to manipulate symbols — the hallmark of human thought. As microcomputers increase in power, AI technology is showing up in games and educational software.

Creating a Game with Logo

To create a simple program for the game of Pick Sticks, you must first define an overall procedure. As shown at right, the main procedure, TO PICKSTICKS, contains a number of subsidiary procedures. Beginning with TO RULES, each subprocedure must be defined before the game will work.

(The procedures TEXTSCREEN, CLEARTEXT and MIXEDSCREEN are Logo "primitives," basic procedures built into the language; you need not define them.)

```
TO PICKSTICKS
RULES
PLAYERS
SETUP
PLAY
END

TO RULES
TEXTSCREEN CLEARTEXT
PRINT [* * THE PICK STICKS GAME * *]
PRINT [] PRINT []
PRINT [DO YOU WANT TO READ THE RULES?]
PRINT [( PLEASE TYPE Y OR N )]
IF RC = "N [STOP]
PRINT [] PRINT []
PRINT [YOU START WITH 15 STICKS.]
PRINT [YOU MAY PICK UP 1, 2, OR 3]
PRINT [STICKS AT A TIME.]
PRINT [THE PLAYER WHO PICKS UP THE]
PRINT [LAST STICK IS THE WINNER!]
WAIT 200
END
```

The next subprocedure that needs definition is TO PLAYERS, to get the names of the two players. This is followed by a definition of the TO SETUP procedure, to set 15 colored sticks in position on the screen. Drawing the sticks is a complicated procedure in itself, so in planning SETUP, simply call this procedure DRAWSTICKS. Then, in the next step, define TO DRAWSTICKS.

```
TO PLAYERS
TEXTSCREEN CLEARTEXT
PRINT [THIS IS A GAME FOR 2 PLAYERS.]
PRINT [FIRST PLAYER'S NAME?]
MAKE "PLAYER1 READWORD
PRINT []
PRINT [SECOND PLAYER'S NAME?]
MAKE "PLAYER2 READWORD
END

TO SETUP
CLEARSCREEN MIXEDSCREEN DRAWSTICKS
MAKE "REMAINING 15
END

TO DRAWSTICKS
PENUP SETPOS [-130 10]
REPEAT 15 [SETPC 1 + RANDOM 3 STICK RIG→
HT 90 FD 18 LEFT 90]
SETPOS [-130 10]
END
```

Once you have learned the principles of turtle graphics, you can experiment with writing simple programs for such familiar games as Pick Sticks *(above and opposite)*. Before beginning, follow the instructions in the Logo manual for making a file disk so that you can save your game program when it is completed.

The first step in planning a Logo program is to divide the project into its most obvious segments or procedures. In the case of the game illustrated on these pages, the overall procedure is called PICKSTICKS; the segments might be: giving the RULES of the game, getting the PLAYERS' names, SETTING UP the screen and starting to PLAY. After naming these key procedures, you must then define them. The definition may include simple steps like FORWARD 20 *(pages 52-53)*; it may also include subsidiary or subprocedures, which must be defined. (Procedures that contain other procedures are sometimes called supraprocedures.)

Each procedure is thus a self-contained module, and you can make revisions without having to redo large portions of completed work. Instead you can modify subprocedures or simply add on new ones. In the program shown here, for example, you could add a procedure to make the computer play a tune after each pickup without disrupting the procedures that let you play against another player.

The subprocedure TO PLAY is also defined in terms of other subprocedures: TELLPLAYER, which will give the name of the next player; GETNUMBER, which will ask how many sticks to take away; CHANGESTICKS, which will erase the number of sticks specified in GETNUMBER; and WINNER, which will announce the winning player when the last stick is removed. The definitions of the subprocedures TO TELLPLAYER and TO GETNUMBER are shown on the screen at right.

```
TO PLAY
TELLPLAYER
GETNUMBER
CHANGESTICKS
IF :REMAINING < 1 [WINNER STOP]
PLAY
END

TO TELLPLAYER
MAKE "PLAYER :PLAYER1
MAKE "PLAYER1 :PLAYER2
MAKE "PLAYER2 :PLAYER CLEARTEXT
(PRINT SENTENCE [OK,] :PLAYER [, IT'S Y→
OUR TURN!])
END

TO GETNUMBER
PRINT [HOW MANY STICKS WILL YOU PICK UP→
?]
MAKE "NUMBER READCHAR PRINT :NUMBER
END
```

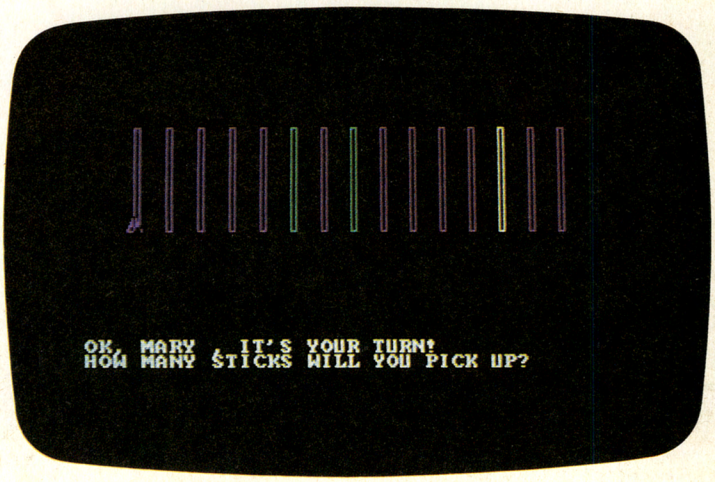
The subprocedure TO CHANGESTICKS erases sticks from the screen by telling the turtle to SETPC 0 STICK, meaning "Use black (0) as the pen color (PC) to draw sticks." Using black makes whatever the turtle draws invisible on the screen. STICK, a key procedure used in both CHANGESTICKS and DRAWSTICKS, is defined here. Last, defining TO WINNER will produce the name of the winning player. Save the game onto a file disk, under the filename *game*.

```
TO CHANGESTICKS
MAKE "REMAINING :REMAINING - :NUMBER
IF :REMAINING < 0 [MAKE "REMAINING 0]
REPEAT :NUMBER [SETPC 0 STICK RIGHT 90 →
FORWARD 18 LEFT 90]
END

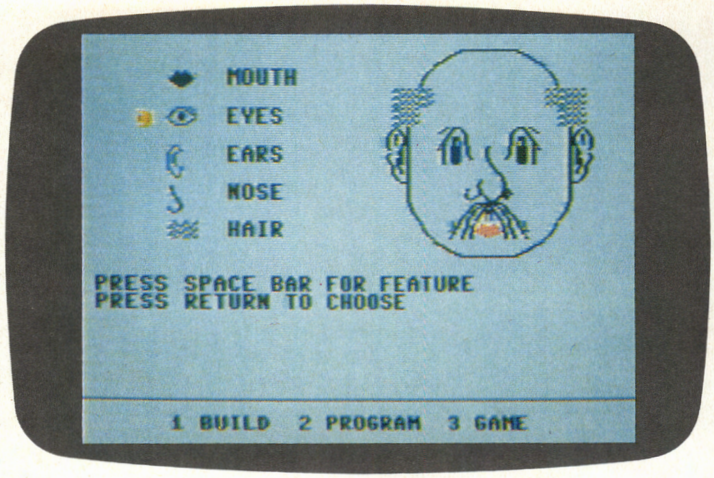
TO STICK
PENDOWN
REPEAT 2 [FORWARD 70 RIGHT 90 FORWARD 3→
 RIGHT 90]
PENUP
END

TO WINNER
PRINT [AND THE WINNER IS ...]
WAIT 100
PRINT :PLAYER
WAIT 100
END
```

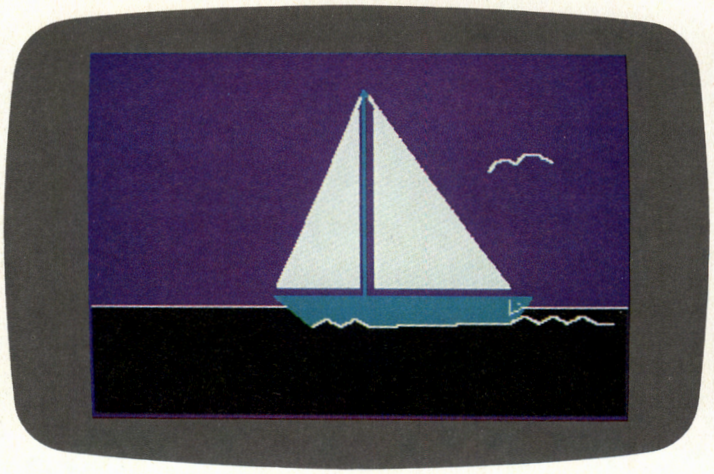
With the game program saved, all you need do to play the game is load Logo into your PCjr, insert the file disk and call up the game by typing **LOAD "GAME**. When the *?* prompt appears, simply type **PICKSTICKS**. The program will show you the rules if you want them, and then will display the opening set of sticks as shown at right.



An Artful Approach to Learning and Fun



With interactive software like this face-making program, even four-year-olds can get an introduction to the keyboard and to programming. The child selects various features to create a face, then animates it by typing a one-letter command such as **W** for wink, **F** for frown. At the next level, the computer makes the face make faces — smiling, winking, frowning in a given order; the child must remember the sequence and type the appropriate letters.



Computer novices of all ages can use the program shown at right to create video art on a color monitor while also learning simple computer programming. Easy-to-remember commands, such as **D** for draw, **R** for right turn and **E** for erase, guide a delta-shaped cursor to draw and paint on the screen. The program also features a special mode of drawing that turns the single pointer into four pointers, enabling you to produce images with four-way symmetry.

For the first-time computer user, child or adult, graphics and animation programs offer an attractive combination of practicality and fun, a way to learn the computer keyboard and basic concepts of programming almost without realizing it. With simple one-key commands, you can produce a picture by selecting items from a menu *(above)* or by creating it out of your own imagination. With several such commands, you can build a simple animation *(top)* or a color chart. Many of the best programs will grow with you, as you progress from one-step commands to more complex instructions.

If you dream of remodeling your home or landscaping your yard, some graphics programs will let you design a project on the screen first. These programs easily produce the straight lines, right angles, curves and circles that enable you to draw floor plans or tree and shrub icons. Mistakes cost nothing to make and are easy to correct. For the business person, some graphics packages mix text and graphics to let you create precise charts and graphs.

To find the best programs for your needs, consult any of the periodicals and journals devoted to hands-on software reviews. You should also talk to friends or check the software best-seller lists in computer stores.

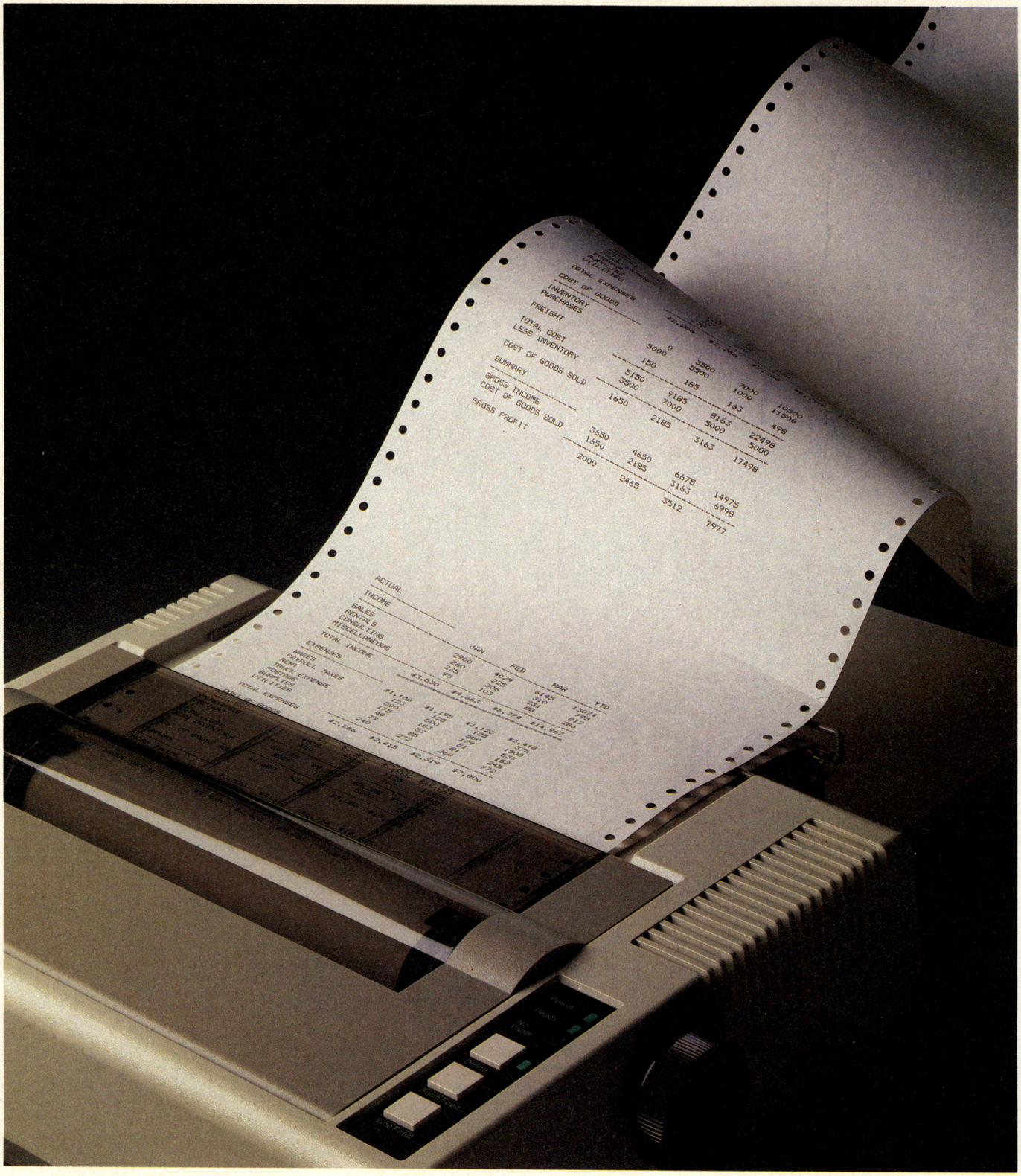
Pairing an Artist's Pad and the PCjr



Of the roughly half-dozen ways to create and handle graphic images on a computer video screen, the touch pad *(above)* is probably the most natural: Using it is as simple as drawing on paper with a pencil.

Once attached to the PCjr system, the touch pad should be set on a hard flat area, with its work surface facing up. Making strokes on the work surface with a plastic stylus or your finger creates lines or broad strokes, not on the pad but on the computer screen.

The best touch pads come with well-designed programs that offer a variety of commands. Typical commands let you draw freehand, execute and adjust precise geometric shapes, save a picture or call up a help menu. Some programs also offer patterns and colors that may be used to fill in, or "paint," designated areas of the screen.


COST OF GOODS
INVENTORY
PURCHASES
FREIGHT
TOTAL COST
LESS INVENTORY
COST OF GOODS SOLD
SUMMARY
GROSS INCOME
COST OF GOODS SOLD
GROSS PROFIT
ACTUAL
INCOME
SALES
RENTALS
CONSULTING
MISCELLANEOUS
TOTAL INCOME
EXPENSES
JAN
FEB
MAR
YTD
WAGES
PAYROLL TAXES
RENT
PRINTING EXPENSE
SUPPLIES
UTILITIES
TOTAL EXPENSES
POWER
READY
ON LINE
CONTROL
SELECT

Programs for Productivity: Managing Your Home or Office

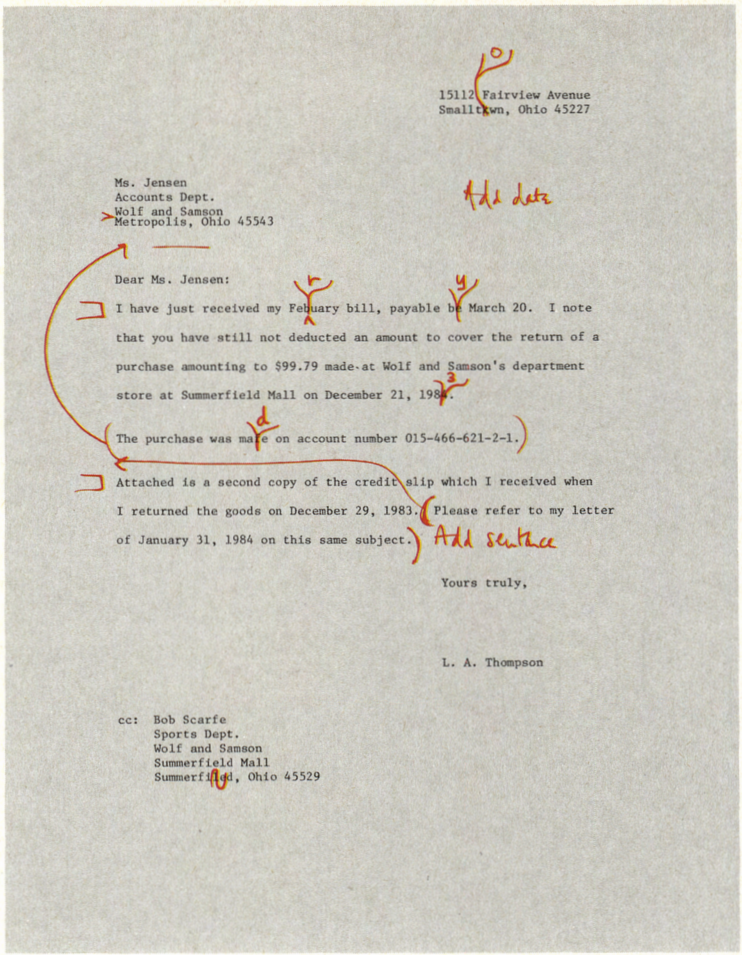
Well-designed software packages are the tools that make your IBM PCjr into the nerve center of a highly efficient small office. Word processing programs can turn your computer into a supertypewriter and you into a speedier typist, versatile editor and champion speller. Electronic spreadsheets — programs patterned after ruled ledgers used by accountants — let you ask "what if" financial questions: Once you have entered the mathematical relationships between various financial elements into the spreadsheet, you can experiment with alternate figures and the program will make all the recalculations for you. Some spreadsheet programs also allow you to turn numbers into graphics, cutting the time needed to prepare high-quality charts and graphs from days and weeks to hours or minutes. Data base management systems can help you store and keep track of entire filing cabinets full of information — and find any individual item in a trice.

For each type of application, software can range from the basic to the elaborate — with price tags to match. Finding the best software for your needs takes a bit of homework on your part. If you are thinking about purchasing a program to help you at tax-filing time, for example, make sure that the manufacturers have a reasonably priced plan for updating your software each year. There are also versatile programs, called integrated software, that offer more than one application within a program. But beware of inflated claims; sophisticated features found in individual software may be sacrificed to save memory space in an integrated package.

Equally important is whether the software is easy to learn and use. Pay close attention to the documentation, or written material, that comes with the program, and to onscreen aids, if any. Good software requires thousands of hours of programming time to create, but you should not have to spend thousands of hours mastering its commands and options. See the software demonstrated and try it out yourself before purchase. For some applications, you will need additional memory, for which you will have to install an expansion board in your system unit *(pages 98-99)*. Finally, the application program must use the operating system of your PCjr; you may have to copy command and control programs from your DOS disk onto the application disk in order to make it self-loading.

On instructions from the computer, a printer puts out a paper, or hard, copy of an accountant's worksheet created by a special program called a spreadsheet. With the help of similar application programs, you can do many jobs, such as financial planning, formerly left to specialists.

Writing and Editing Electronically



15112 Fairview Avenue
Smalltown, Ohio 45227

Add date

Ms. Jensen
Accounts Dept.
Wolf and Samson
Metropolis, Ohio 45543

Dear Ms. Jensen:

I have just received my February bill, payable by March 20. I note that you have still not deducted an amount to cover the return of a purchase amounting to $99.79 made at Wolf and Samson's department store at Summerfield Mall on December 21, 1983.

The purchase was made on account number 015-466-621-2-1.

Attached is a second copy of the credit slip which I received when I returned the goods on December 29, 1983. Please refer to my letter of January 31, 1984 on this same subject.

Add sentence

Yours truly,

L. A. Thompson

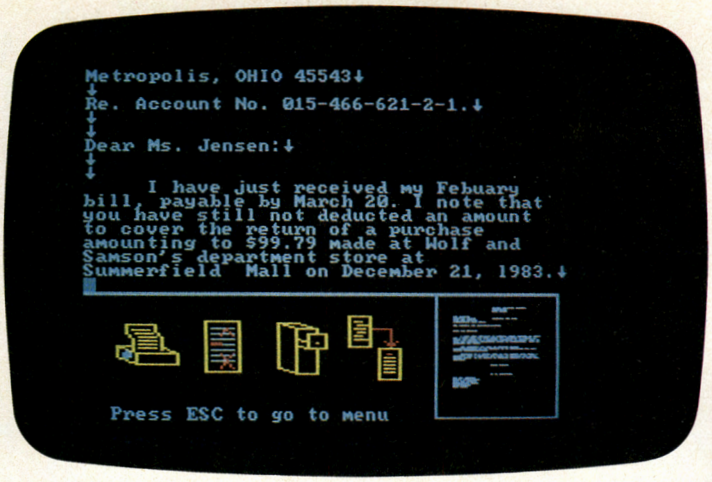
cc: Bob Scarfe
Sports Dept.
Wolf and Samson
Summerfield Mall
Summerfield, Ohio 45529

With a typewriter, any changes to this letter — the spelling corrections, word substitutions and format changes marked in red — would require you to retype the entire letter on a clean sheet of paper. Further mistakes would have to be corrected on the new sheet — or in yet another retyping. With a word processing program, the document is drafted and corrected on the computer screen *(opposite)*; not a word is committed to paper until the entire letter is ready.

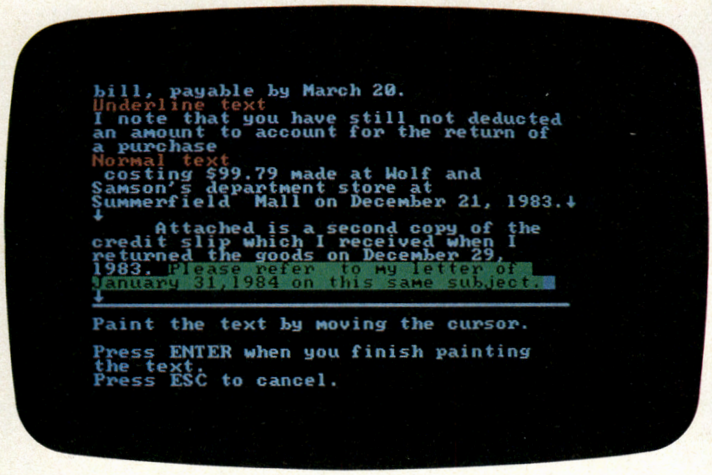
The PCjr adapts readily to use as an electronic typewriter, one of its most valuable functions. With word processing software and a printer, you can create documents as casual as a shopping list or as complex as a business report or book manuscript. Any word-processing software will equip you to edit, format, store and retrieve your text.

To turn out sophisticated reports, however, complete with page headings, underlining, footnotes and aligned tables, you will need more advanced software. To send form letters requires a program that combines a list of names and addresses with a letter file, and then prints the same letter for different recipients. But elaborate programs may be difficult to master. When shopping, look carefully at onscreen aids, or menus; these can be so clear that they practically lead you by the hand — or they can be cryptic and confusing.

In some programs, the menus occupy a constant portion of the screen, providing a handy list of keys for various commands; in others, the menus may be called up when needed. A formatting menu, for example, might list the commands to change margin settings and linespacing. A menu for manipulating blocks of text might tell you what commands to use to incorporate material from one file into another, to mark text you wish to delete or move *(opposite)* or to save the file you are working on.

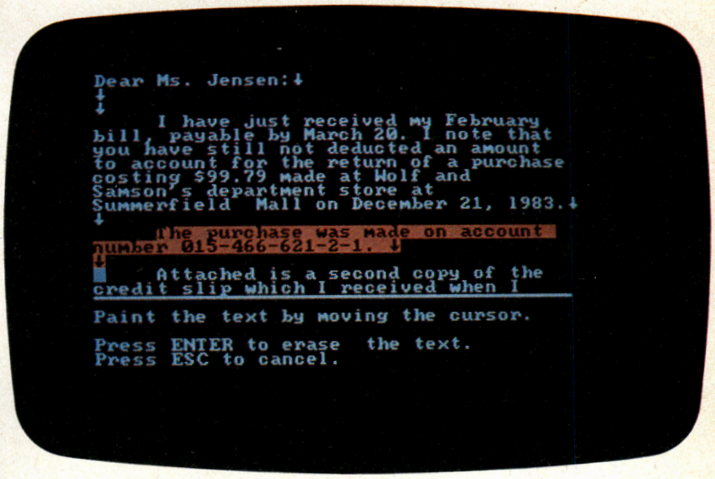


Initial editing indents paragraphs and corrects typos. Screen icons display program options, and the box at lower right shows the completed document as it would print out on standard 8½-by-11-inch paper — useful if you are editing on a 40-character screen.

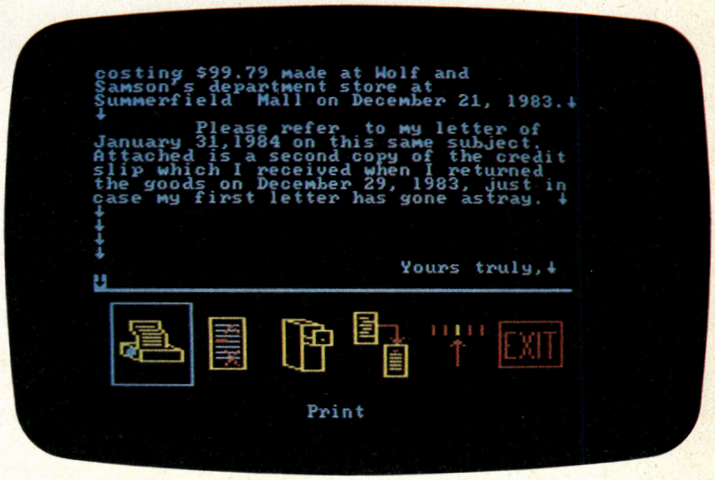


In continued refining of the text, special instructions designate an important sentence to be underlined, a feature of many word processing programs. The cursor has also blocked out another sentence in order to move it to a different part of the letter.

Further editing then corrects spelling errors, and marks the second paragraph for erasure. Some programs let you retrieve erased text if you change your mind.

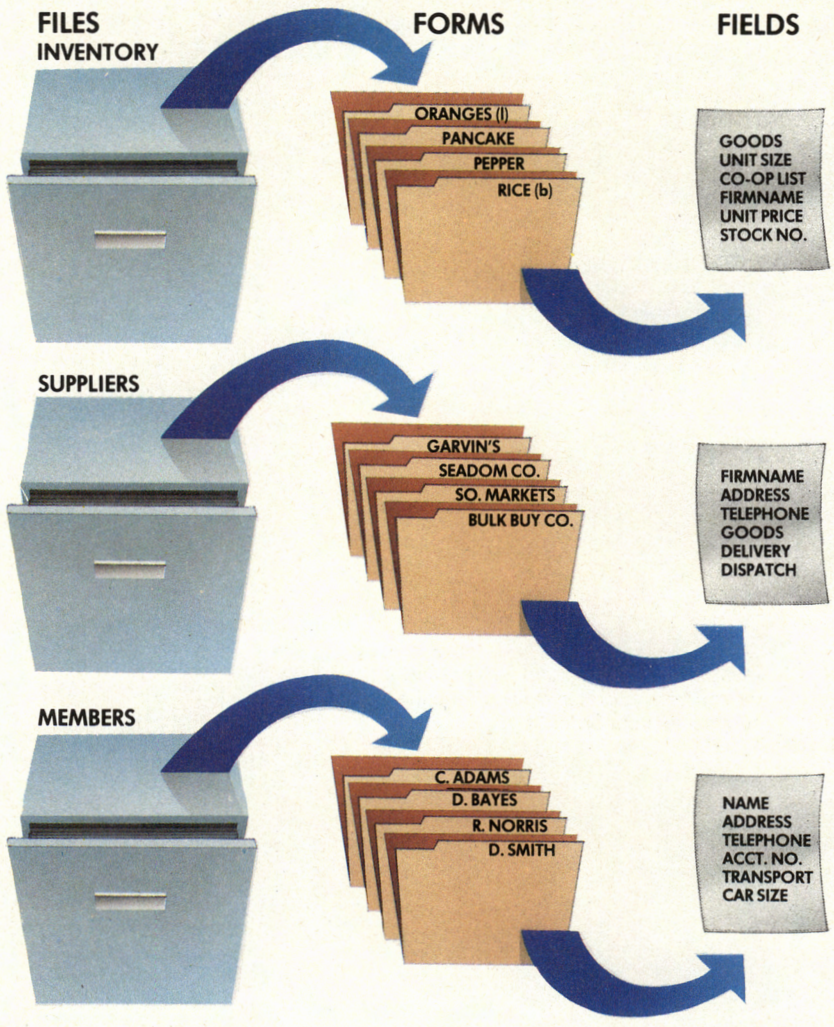


The marked sentence has been moved and the text is instantly reformatted. Most programs allow you to shift designated text by moving the cursor to the new location and giving the insertion command. With editing complete, the letter is ready to print.



Data Base Managers: Electronic File Cabinets

Data base managers can help you organize large quantities of information for storage on, and quick retrieval from, cassette or disk. The information is stored in data files, which are subdivided into forms and fields. Fields are categories of information with labels such as STOCKNUMBER or TELEPHONE. A collection of related fields makes up one form; all forms in a file have the same fields, holding different information. If you designate one field as the key, you can arrange forms in alphabetical or numerical order, or tell the computer to search other files for additional information.



With a data base system and an 80-column display monitor, you can transform the PCjr into an electronic filing cabinet capable of maintaining records for a neighborhood food cooperative, compiling results of a community survey or keeping an inventory of acquisitions for a serious collector. Far more powerful than any traditional filing system, however, a data base manager not only stores and retrieves information with extraordinary speed, but it sorts through and presents whatever portions of it you request.

You enable the PCjr to perform these feats by entering information in distinct categories on a form (sometimes called a record), which you design. If you use your data base to maintain your Christmas mailing list, for example, the form you design for the file might include the following entries: NAME, ADDRESS, CITY, STATE, ZIP CODE and TELEPHONE NUMBER. Each entry is known as a field; all of the forms in a file have identical fields. The best data base systems make it easy to restructure and edit your files and to add, delete, move or change a field when you wish.

A typical program may provide for up to 80 characters per field and up to 254 characters per form. A total of 2,200 simple forms, divided among a number of files, can be stored on a 320K disk. However, you can keep each file on a separate disk, to leave room for expansion.

The fields on this inventory form used by a food co-op include the name of a firm selling oranges, the quantity available and the price. Using such information stored in your computer, you can shop for the best buy on oranges for the co-op. First, you direct the computer to search for "Oranges" in the DESCRIPTION field of all forms in the *Inventory* file. The appropriate records will appear on the screen, one by one.

```
Description: Oranges (m)     Firmname: Southern Markets

Unit Size: Box (88)          Unit Price: $6.50

Co-op List No: 5             Stocknumber: 53/88

At Co-op Store: 0

------------------------------------------------------

File: Inventory        Form 15              Page 1
F-2 Print Form                        F-3 Remove Form
F-5  Date              F-6  Time       F-10  Continue
```

You might then want to find out which of the firms supplying oranges will also deliver them. You can instruct the computer to search the *Suppliers* file for two pieces of information: "Citrus Fruit" in the GOODS field and "Y" — for yes — in the DELIVERY field. By comparing both cost and delivery information from your files, you can find the best supplier for your order.

```
Firmname:                    Goods: Citrus Fruit

Contact:

Address:                     Delivery: Y

                             Dispatch:

Telephone:                   Firmcode:

------------------------------------------------------

File: Suppliers        Retrieve Spec          Page
F-5 Date               F-6 Time          F-10 Continue
```

If no citrus supplier offers delivery in time for your needs, you can look for another solution. By directing the computer to search the food co-op's *Membership* files, keying on the CAR SIZE and TRANSPORT HELP fields highlighted on the form retrieved at right, you could find a co-op member both equipped and available to pick up the order.

```
Name: Mary and Michael BROOKS          Car Size: L

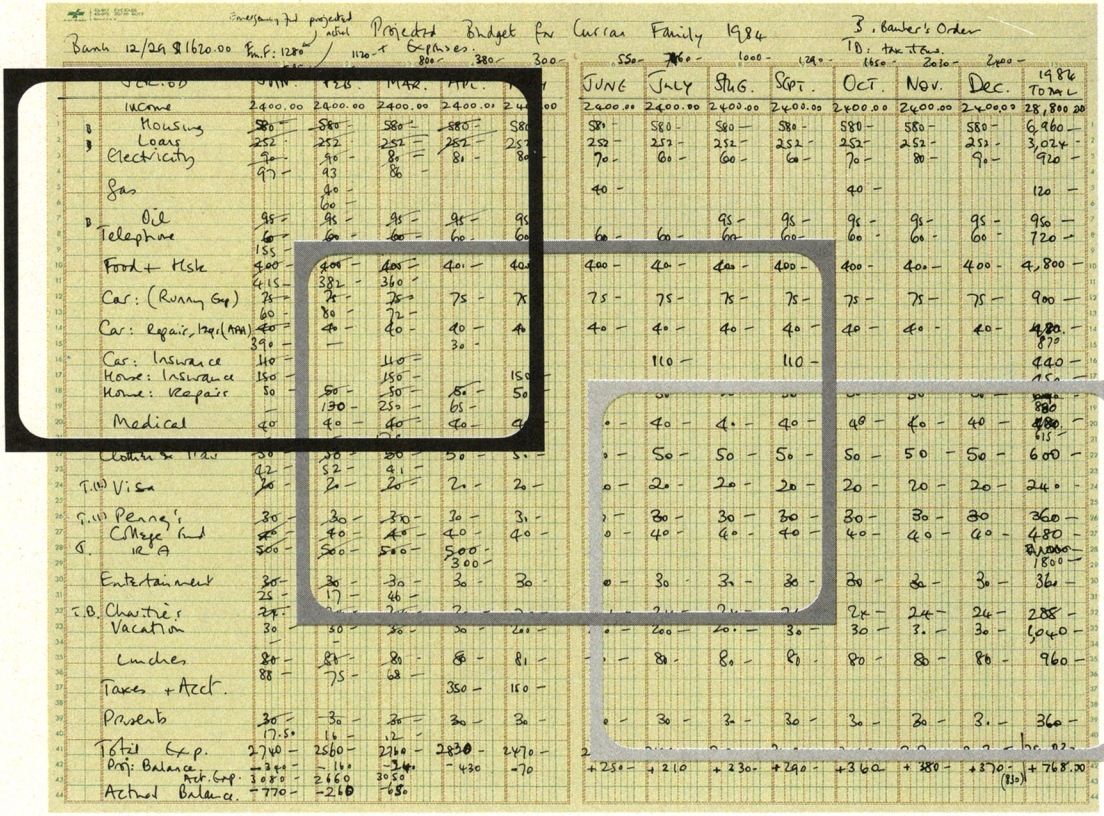
Address: 34 Evans Court                Transport
         Rolling Road                  Available: FRI (p)
         Smalltown,
         OHIO 45333
                                       Account No: 32
Telephone: 614-555-9787

No. in household: 5                    Basics: Y

------------------------------------------------------

File: Members          Form 32                Page 1
F-2 Print Form                        F-3 Remove Form
F-5 Date               F-6 Time         F-10 Continue
```

Easy Recalculation with Spreadsheets



Computer screens overlaid on a paper worksheet *(above)* illustrate the window effect of a spreadsheet program. Unlike a paper version, the electronic model can have hundreds of columns and rows without growing impossibly unwieldy. By scrolling or by using direct **Go to** commands, you can call any section of the worksheet into view on your screen.

A spreadsheet program lets you build a model of the accountant's traditional worksheet on your computer screen. You can use such a program to prepare budgets, to record expenses, to compute taxes, to manage your stock portfolio.

Like its paper counterpart, the electronic model is essentially a blank page divided into rows (usually numbered, from top to bottom) and columns (usually lettered, from left to right). Columns and rows intersect to form cells, which are identified by their coordinates. An electronic spreadsheet can have more than 10,000 separate cells. The cells can hold different types of information: text, such as labels for columns and rows; numbers; and instructions in the form of mathematical formulas.

Formulas can be simple, such as an instruction to add all the numbers in a certain range of cells and put the total in another cell; or complex, such as an instruction to perform a given calculation *if* a specified condition exists. Once the formula is entered, any changes in the cells it uses in its computations automatically produce changes in all cells where the formula was applied. With a spreadsheet program, such a change takes only a few seconds, instead of the hours it might take by hand. This lets you try different "what ifs," to see how variations in one area, such as monthly income, affect the numbers in the rest of the spreadsheet.

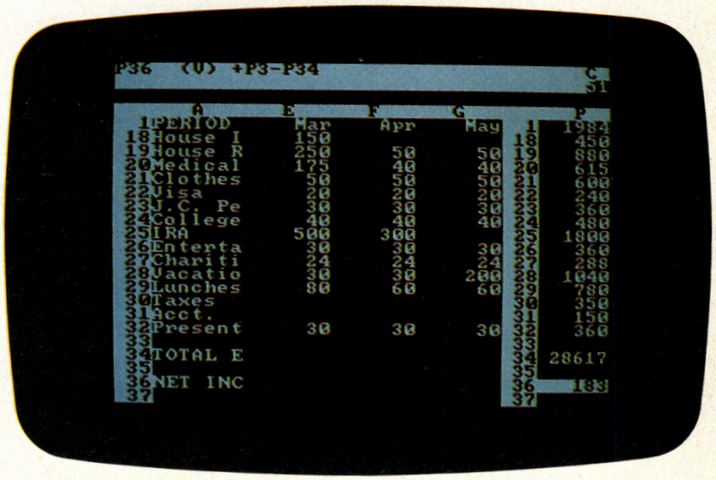
A typical spreadsheet screen displaying a projected household budget reserves one or more lines at the top for display of program-related information. The top line identifies the cell — N19 — where the cursor is located; (v) +M19 indicates that the program has entered a value determined by previous instructions — in this case, the value from cell M19. The next two lines show your command to move the cursor to cell J12 without scrolling the screen.

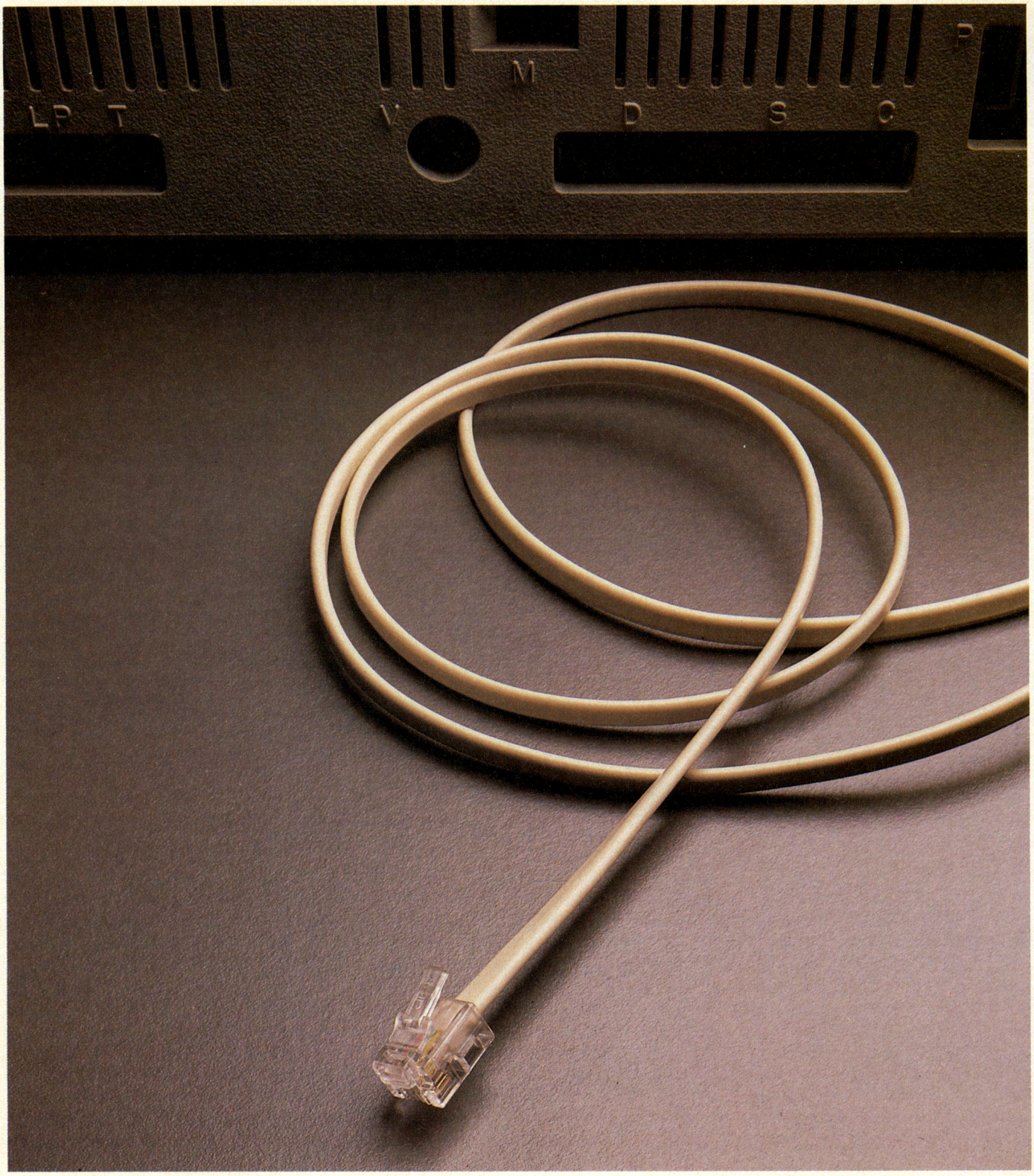


A formula instructs the computer to display the result of a calculation that uses figures from other cells. At the top of the screen at right, the program information tells you that the highlighted figure in cell P20 is the sum of all the values in cells C20 through N20. A tally of actual expenditures in Column P reveals that an unexpected increase in house repair and medical expenses (added up in P19 and P20) have thrown your projected budget into the red.



A convenient feature of most spreadsheet programs is the split screen, which lets you view different parts of your worksheet simultaneously. You can change material in one place and almost instantly see the effect elsewhere. By reducing your April IRA payment and lunch expenses all year, as displayed on the left half of the screen, you will get the beneficial effect — a balanced budget — that appears on the right.




LPT
M
D
S
C

Computer Communications

The potential of your PCjr is greatly increased by its ability to "talk" with other computers, using the same telephone network that you ordinarily use for voice communications. Electronic information transfer can be faster than mail and cheaper than messenger.

The advent of technology that adapts telephone lines for rapid data transfer means that your PCjr can be linked to many different types of computers, perhaps thousands of miles away. And the PCjr can draw on the capabilities of those remote computers to do jobs it could not do on its own. For instance, your PCjr can talk with mainframe computers that control huge data bases, which you can read and use for your own purposes. You can do research in a specialized data base such as a computerized law library or an electronic newspaper.

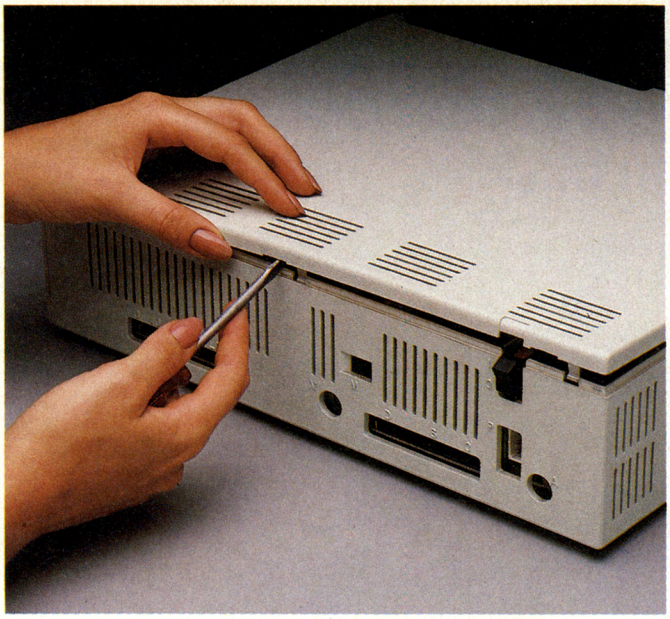
An easy way to get access to a variety of data bases is to subscribe to a commercial information service, a computer network that draws on the facilities of a number of data bases. After making the phone link with the service, you can leaf through its menus electronically until you reach the subject you want to work on, then track down the specific items that interest you. You can even set up your computer to record this information, as hard copy from your printer or as a disk file.

Research data bases aren't the only offerings of an information service. Some have a wide variety of features, from movie reviews to constantly updated news services. In some cities, you can even get restaurant menus and theater schedules through an information service. And several services have facilities that let subscribers communicate with each other, by notices posted on electronic bulletin boards, by individually addressed messages, or even by direct conversations in which words entered at one keyboard may appear on a monitor hundreds or thousands of miles away.

But you don't need an information service to conduct direct communications. Any two personal computers can be set up to talk to each other over the telephone lines; you can send reports from office to office, or electronic mail from city to city. Some computer owners even use their equipment to set up small-scale bulletin boards. Whatever the aim of your communications, you will find a vast pool of computerized information available to you — quickly and easily.

A modular phone jack, like the kind that lets you unplug a telephone, allows you to plug your PCjr into the telephone network. The jack is attached by a cord to a direct-connect internal modem, which translates the electronic pulses of computer-encoded data into signals that can be sent through the phone lines to another modem. That modem, in turn, converts the telephone signals into pulses intelligible to its computer.

Installing Devices for Telephone Hookups



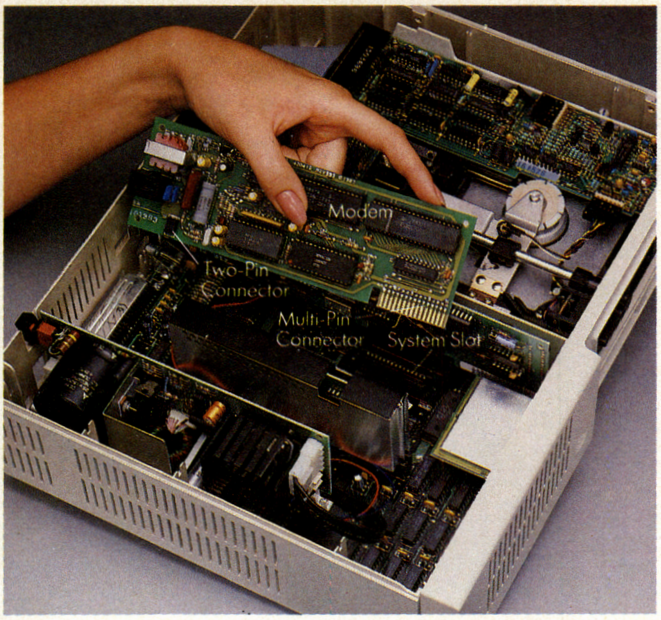
Before installing the internal modem adapter card, first unplug the system unit. Allow the computer to cool for five minutes before removing the cover. Place a flat screwdriver into one of the three slots at the top of the back of the system unit and twist; repeat at the other slots. Lift the cover up and away from the front of the system unit.

A conversation between two people is based on implicit rules that enable them to understand each other. The rules include using speech to communicate, speaking the same language, and taking turns doing the talking and listening.

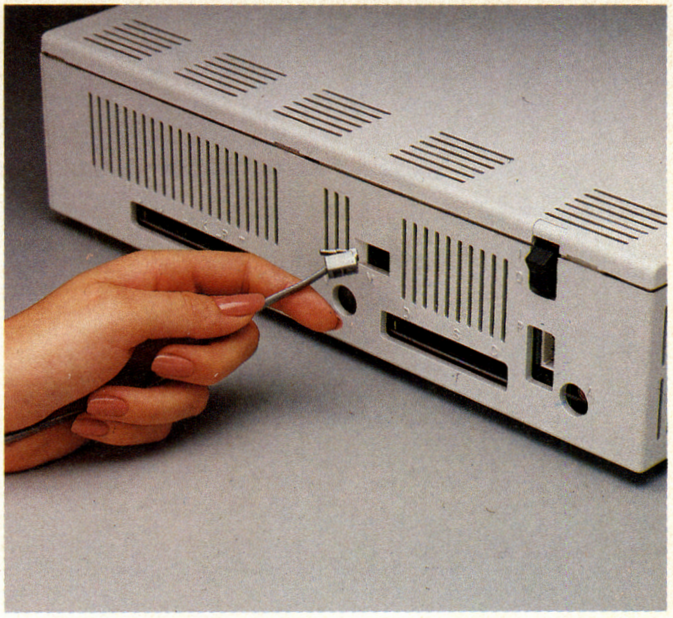
Computers need similar rules to communicate, but because they are much less flexible than people (they can't use slang, for example) their rules must be more precise. To ensure that data transfer can occur without garbling of messages or damage to equipment, computer manufacturers have developed an international communications standard, governing everything from the meanings of electrical codes to the shapes of cable connectors.

The standard also governs the operation of a modem (a short form of **mo**dulator-**dem**odulator), a device used to translate the electronic pulses understood by the computer into signals understood by the telephone system, and vice versa.

With the PCjr, you may use either an internal modem *(above)*, which fits into one of the computer's expansion slots, or an external modem *(box, opposite)*, which plugs into the serial port at the back of the system unit through an adapter cable. An internal modem frees the serial port for other devices such as a printer, and has other useful features: It can dial a number automatically and answer calls coming in to your computer.



To plug the internal modem into the system slot, position the modem with its multipin connector directly above the system slot, and the smaller two-pin connector above the two-pin plug. Press the card firmly in place. Be sure the rear of the card goes into the rear guard.

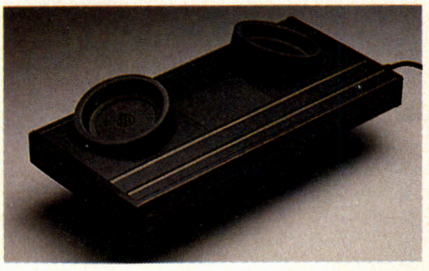


To replace the cover, place it atop the system unit and slide it forward until it is flush with the front of the system unit. Press down on the rear of the cover until it snaps into place. To complete the modem installation, plug one end of the modem cable into the connector slot marked *M* on the back of the system unit. Plug the other end into a standard modular telephone wall jack.

A Modem for Several Machines

An external modem — a separate device that connects the PCjr to phone lines — is generally less convenient than an internal one because it uses the computer's serial port, requiring that you disconnect the printer. But if you have several computers, it is more economical to let them share one modem.

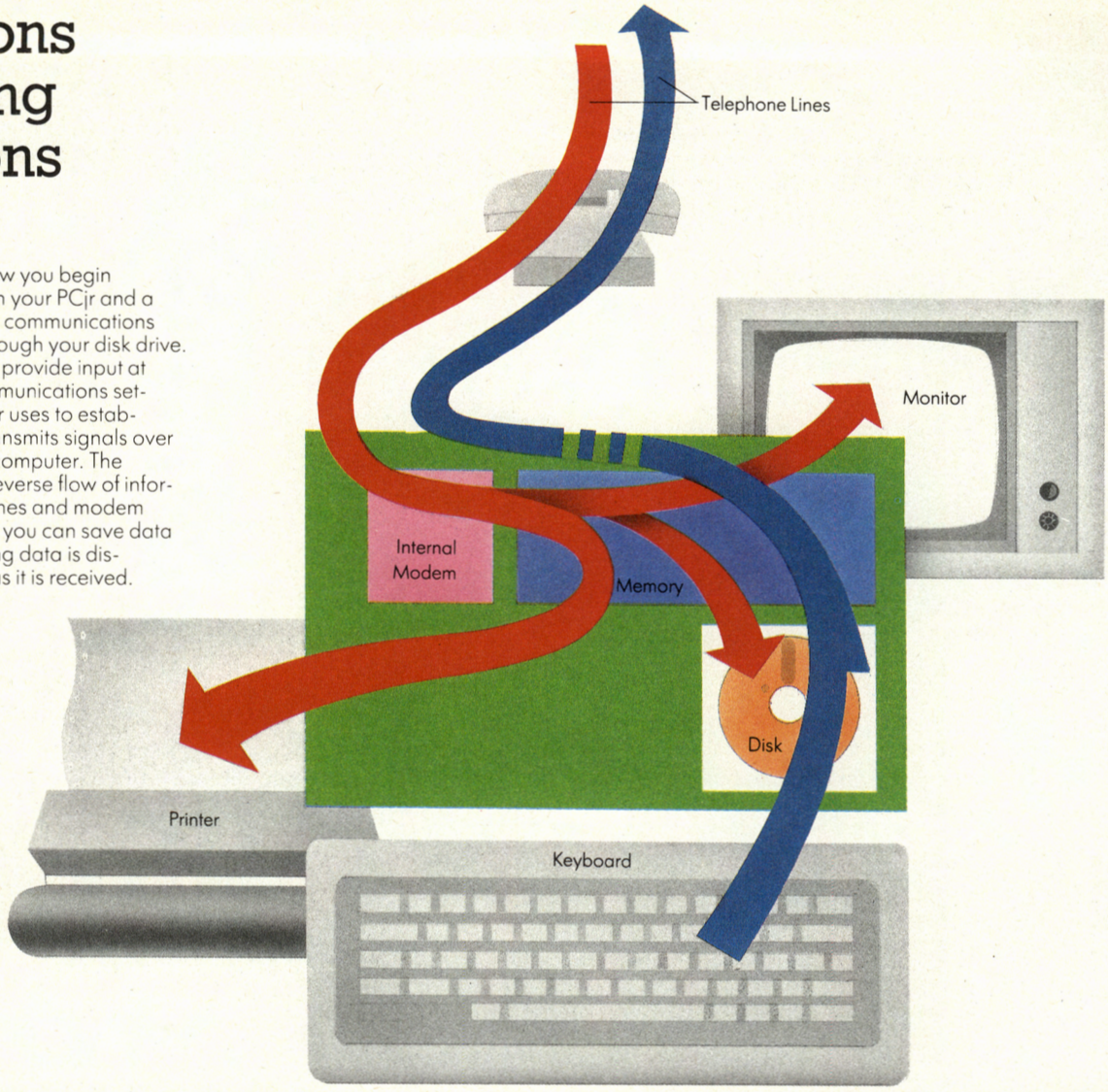
External modems may be direct-connect (plugging into modular jacks like an extension phone) or acoustic. An acoustic modem *(right)* needs no cable connection to the computer: The phone's handset fits into the modem's rubber cups. This is useful if you plan to travel with your PCjr and may be calling from phones without modular jacks.



An acoustic modem turns computer signals into audible tones and sends them via a telephone receiver in the modem's cradle.

Instructions for Linking Operations

The blue arrow shows how you begin communications between your PCjr and a distant computer. Load a communications program into the PCjr through your disk drive. The program asks you to provide input at the keyboard about communications settings, which the computer uses to establish a link. The modem transmits signals over phone lines to a remote computer. The red arrows indicate the reverse flow of information, through phone lines and modem into the computer, where you can save data to disk or printer. Incoming data is displayed on your monitor as it is received.



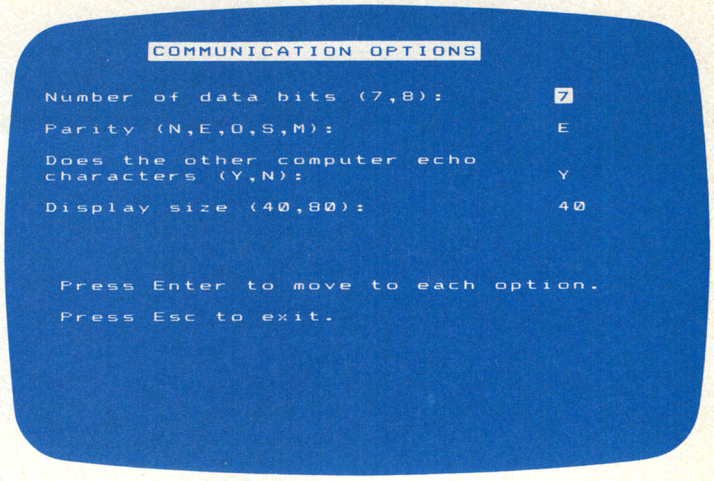
Even with a modem in place, you can communicate with other computers only with proper instructions. A communications program tells the PCjr how to organize its internal operations and how to set up an external link. In effect, this software turns your PCjr into a remote terminal of another computer and directs the communications process.

To assure that both computers will be speaking the same language, you specify protocols, such as baud rate (transmission speed), data bits and the type of parity checking to use in error detection. You also specify whether information will flow in both directions simultaneously (full duplex) or in one direction at a time (half duplex).

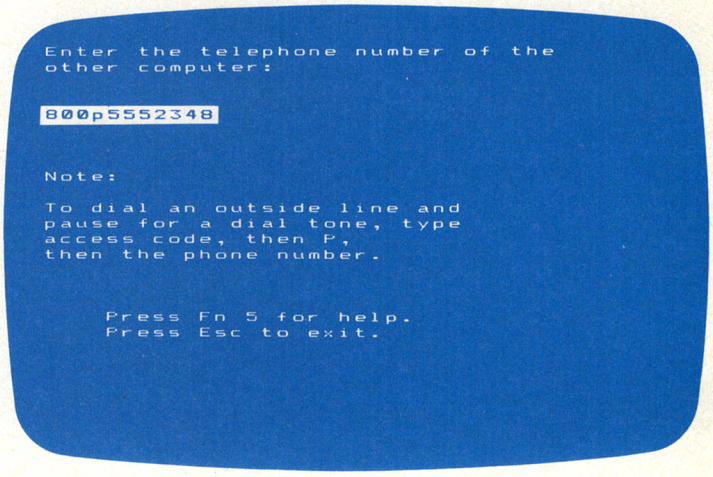
A communications program enables your PCjr to perform a variety of tasks: Your machine can send and receive messages at the moment they are typed, save incoming information on a disk or send information that has been stored on a disk. Such a program can give you access to a mainframe computer, which will enable you to use programs too large to be stored in the PCjr's memory.

Communications programs for the PCjr range from the sophisticated — and costly — to the simple. The Sampler disk that comes with your PCjr contains a simple communications program that lets you immediately link up with other computers and with information services.

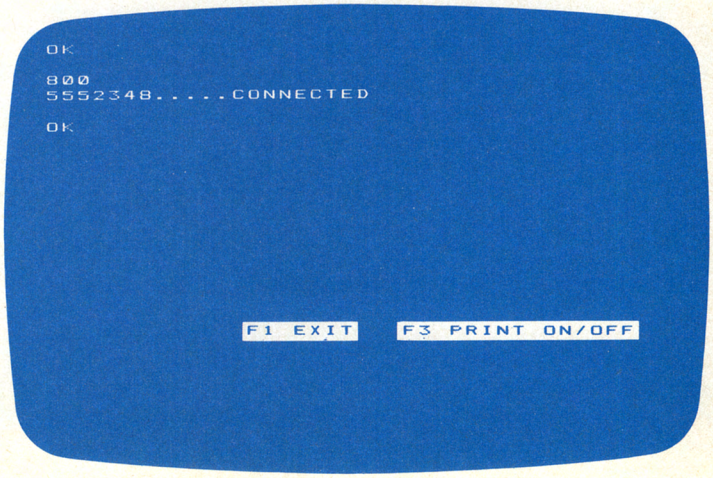
Communications software, like the simple program provided on your PCjr Sampler disk, directs the process of communicating with a distant computer. An options menu *(right)*, asks you to provide information to ensure that your PCjr will be using the same rules, or protocols, as the equipment at the other end of the phone lines.



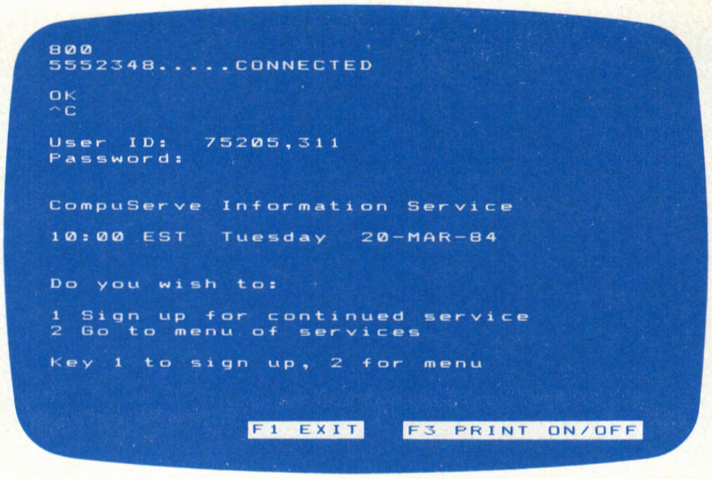
After you've set protocols, another menu *(right)* asks you for the phone number of the computer you want to call. First type an access code — which may be a number to get an outside line, an area code, or both. Type **P** after each code so the computer will pause 10 seconds for the dial tone. Then enter the phone number of the mainframe, personal computer or information service.



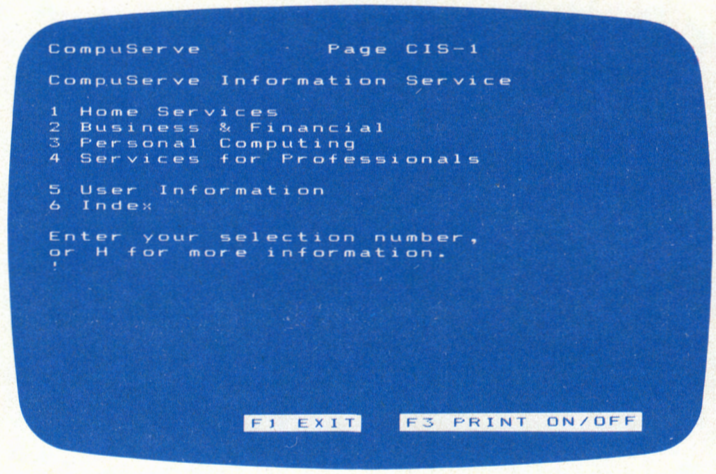
The communications program will instruct the computer and modem to dial the distant computer. Your PCjr will let you know when you've established a link with another computer by displaying a "Connected" message. It will tell you if it has failed to reach the other computer, by giving you a "Busy" or "Unsuccessful" message.



Commercial Services: Information à la Carte



In the first step of a typical procedure for logging on, or tying into a commercial information service *(right)*, you call the service's number and make a telephone connection. After the link is confirmed with messages such as "Connected" and "OK", hold down the **Control** key and press **c**. You will then be asked for your ID and password. To ensure that your password remains private, it is not displayed on the monitor screen when you type it.



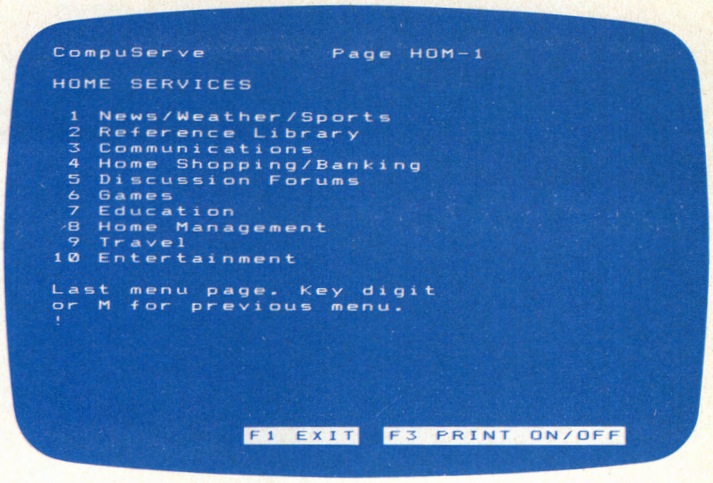
Once you have logged on to an information service, a main menu *(right)* lists an array of services. Each menu entry leads to another, more specialized menu. Entering **2**, for example, will bring up a listing of financial and business services; typing **1** will lead to another menu, listing home services.

One of the most fruitful uses of your PCjr is in linking up with a commercial information service such as The Source or CompuServe. Easy to use, these services place an enormous amount of data at your disposal.

Most information services work in similar ways. You must pay an initial fee (applications are available through computer dealers and by mail), and the service then assigns you an identification number and a password, which give you access to the system. The information service bills you for the amount of time you are connected, at rates that vary according to the time of day. And of course, you pay the telephone company at standard rates for the use of the phone lines. If there is no direct telephone number for an information service in your area, you may be able to use a specialized communications network that allows you to establish a link to a distant service with a local phone call.

The Source and CompuServe not only give you access to a vast array of information instantly, but also offer programs that let you "chat" with other users and hold electronic meetings. You may, for example, choose to join a special-interest group, or SIG, of PCjr users. You can then leave messages, read the messages left by other SIG members, and participate in specially scheduled on-line conferences, which often feature guest "speakers."

The home-services menu *(right)* presents a list of even more specific services. Choosing HOME SHOPPING/BANKING would bring up still another list, which would allow you to leaf through an electronic catalogue and order merchandise, or transfer funds from a checking to a savings account. Once you're familiar with the menus, you can use abbreviated commands to go directly to the service you want.



Saving Time with Special Utilities

Commercial information utilities provide access to a world of information and services. Networks such as CompuServe and The Source offer an assortment of services through one centralized source. Other utilities are more specialized, providing such services as bibliographic searches of professional journals, or detailed information on commodity prices.

These services can save you substantial time and money, by eliminating transportation costs and reducing the time you spend in libraries. But user fees can quickly add up. To keep your costs reasonable, be familiar with user guides before you log on, and have a good idea about what part of the service you want to use and what information you are seeking. Plan to use the service during evenings or weekends, when hourly access charges are lower.

The following is a brief guide to the range of information and services available through your PCjr.

NEWS AND WEATHER: Electronic newspapers range from up-to-the-minute headline services to complete newspapers, including everything from weather and sports to astrology.

FINANCIAL INFORMATION: You can survey constantly changing stock and commodity prices, review data on thousands of corporations, or research current economic indicators.

ON-LINE SEARCHES: On-line reference services let you leaf electronically through reference materials. You can search the *New York Times* index, or an entire encyclopedia, or hundreds of professional and trade publications.

ELECTRONIC PUBLISHING: You can call up special editions of your favorite magazines, or electronically "publish" your own poems, articles, newsletters and books for other users to read.

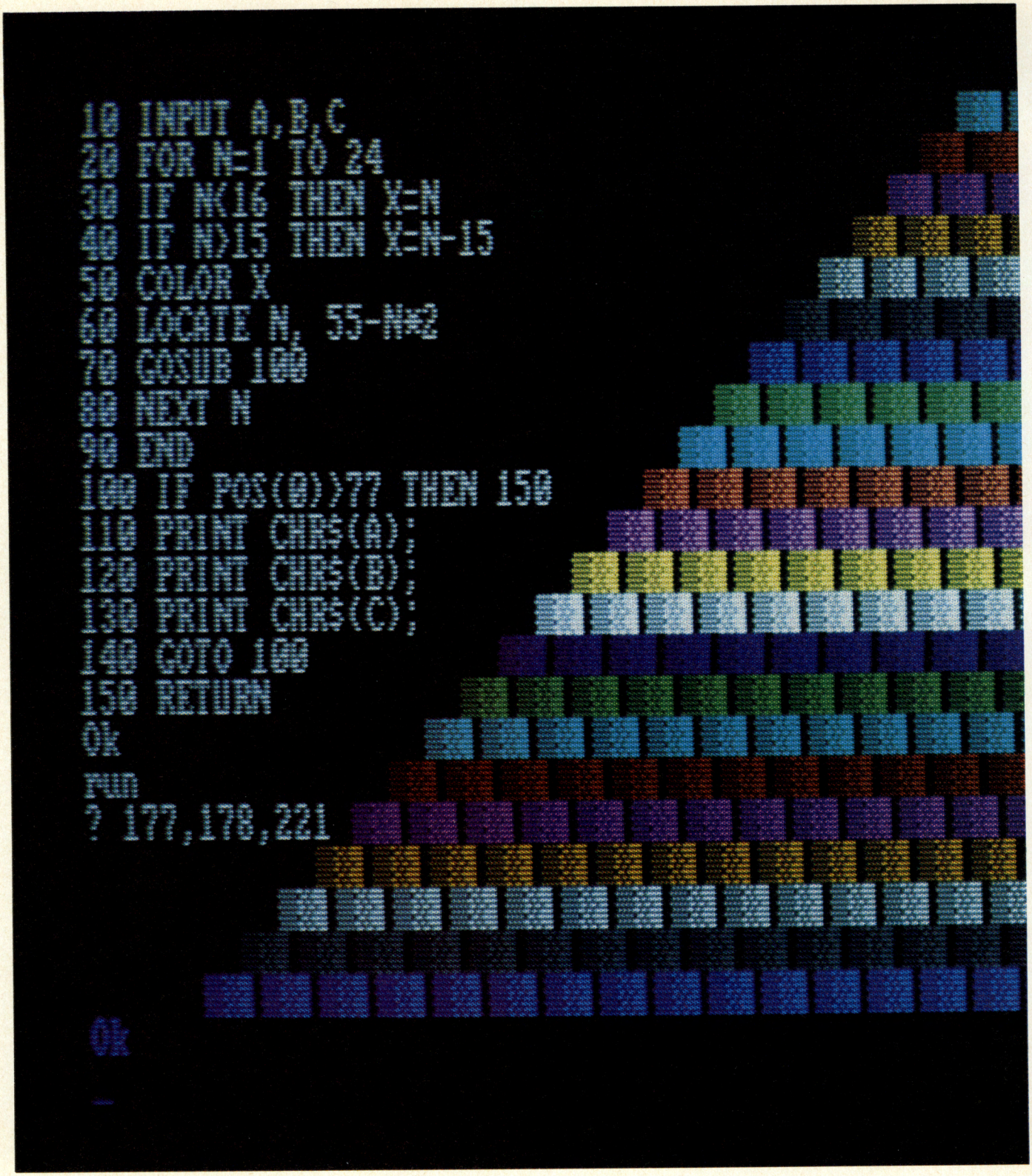
SHOP-AT-HOME SERVICES: Electronic catalogues allow you to do your shopping at home. You can compare prices, place orders and charge your purchases using a credit card number.

BANK-AT-HOME: Computer banking speeds transactions, letting you transfer funds between accounts or instruct the bank to pay bills from your account.

TRAVEL INFORMATION: You can check airline schedules, make reservations, and order and charge your tickets in much the same way a travel agent does.

ENTERTAINMENT: Special listings give you access to movie and theater reviews, restaurant menus, and ticket-ordering facilities. Or you can play multi-user strategy games, chat with friends on a computer version of citizens-band radio, read your horoscope, or use a matchmaking service to line up a Saturday night date.

ELECTRONIC-MAIL: These services save time and eliminate messenger costs by transferring memos, contracts, or letters directly from computer to computer.


10 INPUT A,B,C
20 FOR N=1 TO 24
30 IF N<16 THEN X=N
40 IF N>15 THEN X=N-15
50 COLOR X
60 LOCATE N, 55-N*2
70 GOSUB 100
80 NEXT N
90 END
100 IF POS(0)>77 THEN 150
110 PRINT CHR$(A);
120 PRINT CHR$(B);
130 PRINT CHR$(C);
140 GOTO 100
150 RETURN
Ok
run
? 177,178,221
Ok

Programming in BASIC: Home-grown Software

The job of programming — writing the software that makes a computer so versatile — is often regarded as a task best left to highly trained professionals. But your PCjr is equipped with a programming language, called BASIC, that makes it possible for you to write your own programs, and learn about programming in the process.

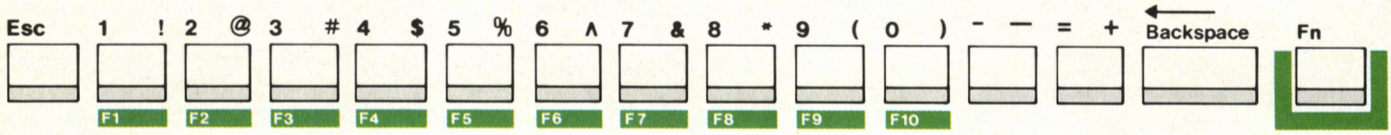
BASIC, like most programming languages for non-experts, is what is known as a higher-level language. Such a language is itself a program that translates easy-to-remember commands — source code — into the machine code that the computer understands. BASIC, which was designed as an instructional language, makes it possible for you to try out different ways of programming and see the results quickly and easily.

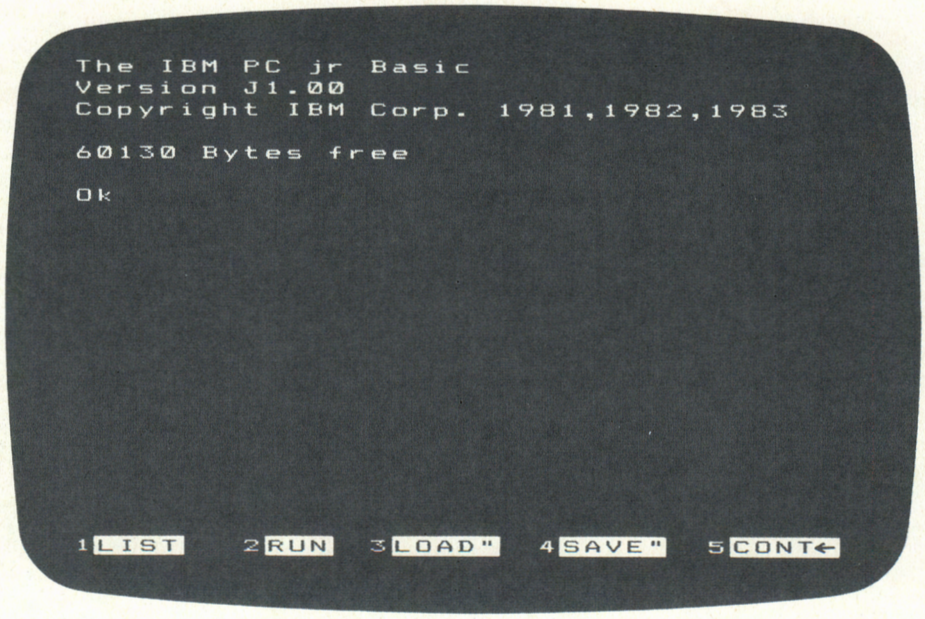
The version of BASIC installed in your PCjr is an interpretive language. Whenever you run a program, BASIC must translate each line of source code into machine code, then execute it, before going on to the next line. While this slows the running of programs, it allows you to detect and correct problems as you develop a program: If it is not doing what you want it to do, you can read and revise the source code. Another type of higher-level language, called a compiler language, translates source code into machine code as it is written. Because compiled programs are already in machine code, they run more quickly than interpreted programs, but the source code is no longer available for examination and change. To combine the advantages of each type, you can use an optional addition to the BASIC package called the BASIC compiler. This program takes the source code of a program that you have developed with the interpreter and uses it to make a compiled program for actual use.

In this chapter you will be introduced to some of the most fundamental commands in BASIC, which let you begin to write simple programs yourself. You should try the examples on your own computer, and try out variations of the examples as well, to get an understanding of the concepts they illustrate. You will also find the *BASIC* manual supplied with your computer to be an excellent reference source as you explore the power of the language. And if you decide you want to go on, you will discover that the popularity of BASIC has given rise to many books and courses that can take you as far into the study and use of the language as you want to go.

A few lines of BASIC program code make the PCjr display this colorful pattern on the monitor. The numbers at the beginning of each line are the backbone of the program; BASIC reads the lines in numerical order, executing the instructions on each line before going on to the next. When this program runs, the shapes within the pattern are determined by numbers typed at the keyboard in response to a question mark on the screen.

Two Modes for Working in BASIC





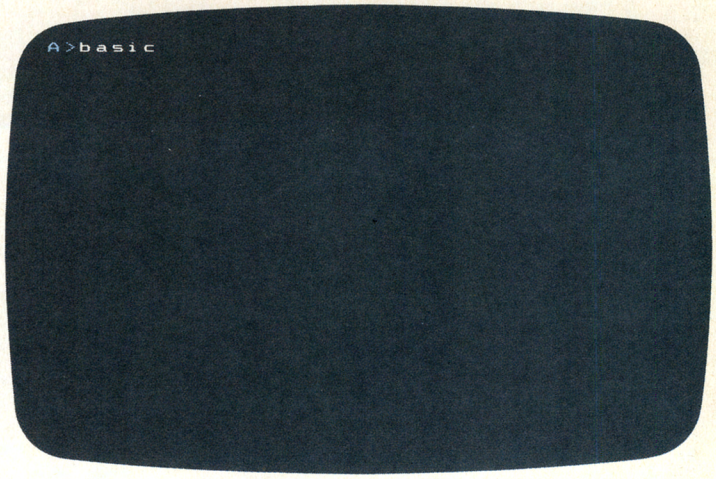
When you start up in BASIC, the first lines on the screen tell you which version of the language you are using. The *Ok* prompt indicates that the BASIC interpreter is ready for your instructions. BASIC lets you enter several of the most common instructions, such as **list**, **load**, **save** and **run**, with just two keystrokes. The blocks at the bottom of the screen show the key words assigned to the function keys at the top of the keyboard. You can clear these key indicators from the screen with the command **key off**; entering **key on** brings them back.

Cassette BASIC and Cartridge BASIC, the versions you can use with your PCjr, differ in their capabilities *(pages 26-29)* but not in the way you interact with them. Cartridge BASIC, which lets you store and retrieve programs on disks, is used in the examples in this chapter. To use Cartridge BASIC with a disk, first put a disk that has the DOS command files *(page 32)* into the disk drive, then start up by turning on the computer or inserting the BASIC cartridge into the slot.

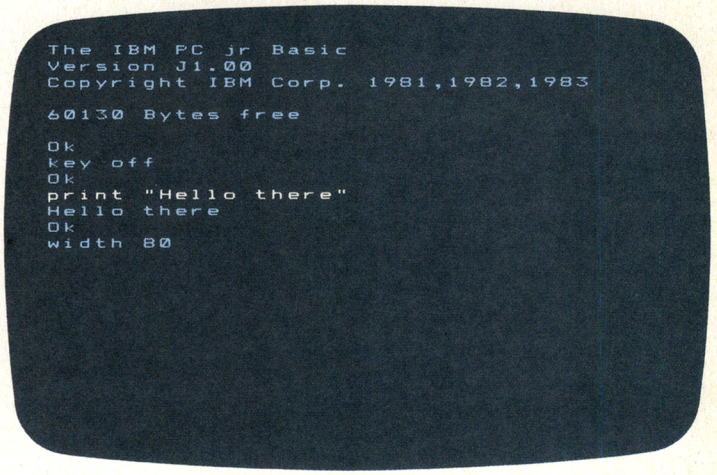
Both versions of BASIC distinguish between immediate and deferred operation. If a BASIC line starts with a character that is not a numeral, the BASIC interpreter understands that immediate action is required as soon as you strike the **Enter** (↵) key. It will analyze your line to see if it makes sense (to BASIC); it will then carry out the instructions and return with the *Ok* prompt, or tell you that the line doesn't make sense, with the message "Syntax error."

If a line starts with a numeral, BASIC understands that you want it to file that line under that number, for later use. When BASIC is operating like this, it is said to be in deferred, or programming, mode (a program is in fact a set of these deferred instructions, filed in memory and executed in numerical order). When you enter a program line, BASIC assumes you intend to continue, and waits for the next line to be entered without giving you the *Ok* prompt.

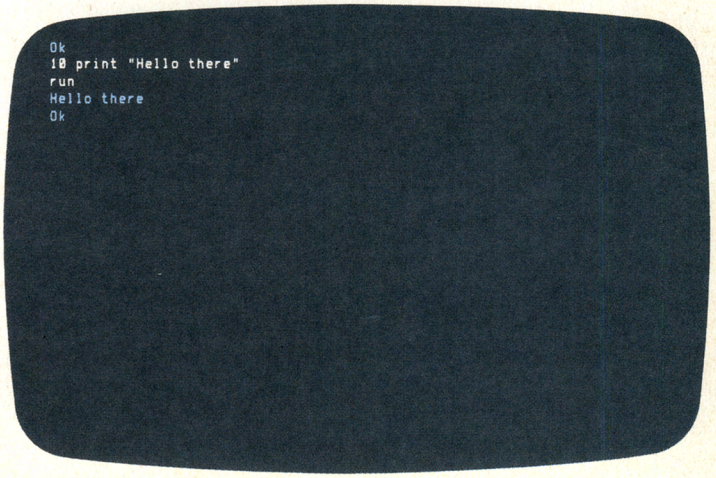
When you are operating in DOS, typing **basic** ↵ causes the computer to turn control over to the Cartridge BASIC interpreter. The BASIC cartridge must be in place or you will see the message *"Cartridge required."* If you do, insert the cartridge into the slot. In either case, you will next see a screen similar to the one shown on the opposite page. In order to return control to DOS from BASIC, type **system** ↵.



Every time you start using BASIC, the language is in the command, or immediate, mode. This means that any instruction you enter will be carried out immediately, as shown at right. When you enter the line **key off** the BASIC interpreter immediately carries out the statement, turning off the key indicators at the bottom of the screen and then signaling with the *Ok* prompt that it is ready for your next instruction. When you enter the line **print "Hello there"** BASIC displays *Hello there* but not the quotation marks. When you enter **width 80**, BASIC changes the display to 80 characters per line, as shown on the screen below.



To change over into the programming mode, simply start a line with a number. When BASIC reads a number at the beginning of a line, it knows that it must store the instructions on that line and carry them out later, when you give it the **run** command. When you press the **Enter** key at the end of a line, BASIC waits for another line number or an instruction. On receiving the command **run**, BASIC executes all of the lines it has stored up, in numerical order of line numbers. In this case there is only one line, and BASIC displays the same message that it earlier displayed in the immediate mode.



Telling BASIC What You Mean

As you enter a program, BASIC keeps it in the computer's internal memory. You can view the program that is in internal memory at any time, using the **list** command *(screen, right)*. To store a program on disk, use the **save** command; give the program a name and be sure to enclose the name in quotation marks. To clear a program from internal memory, enter **new**. As shown on the screen at right, a **list** command shows no program remaining in internal memory after the **new** command is executed.

```
Ok
10 print "Hello there"
20 print "Goodbye now"
list
10 PRINT "Hello there"
20 PRINT "Goodbye now"
Ok
save "greeting"
Ok
new
Ok
list
Ok
```

To move a program from a disk into internal memory, use the command **load** followed by the program name in quotation marks. You can use the **list** command to see the contents of the program, then use the **run** command to see it executed. If you want to empty the screen, enter the command **cls**. Everything on the screen will disappear, and the *Ok* prompt will appear in the upper left corner. The **cls** command affects only the screen, leaving the contents of the internal memory intact, as you can verify by using **list**.

```
Ok
load "greeting"
Ok
list
10 PRINT "Hello there"
20 PRINT "Goodbye now"
Ok
run
Hello there
Goodbye now
Ok
cls
```

Like most interaction with a computer, programming in BASIC requires that you be very precise in formulating your instructions. Each line in a program is made up of two sorts of elements: key words and special characters that have specific meanings in BASIC *(chart, above, right)*, and the ordinary letters and numbers of the data the program processes. These elements must be assembled according to the system (called syntax) that the BASIC interpreter understands. When BASIC encounters instructions that do not make sense within its syntax, it responds with the phrase "Syntax error."

BASIC has a wealth of other messages to tell you when you have made what it considers to be a mistake; they are listed, with their meanings and suggested remedies, in an appendix to the IBM *BASIC* manual.

When you are in immediate mode, BASIC reads each line as you enter it, and responds with an error message if the content of the line is incorrect. When you enter lines in the programming mode, on the other hand, the BASIC interpreter does not check the syntax until you run the program. Then BASIC will tell you what the error is and the number of the line where it occurs, and give you the opportunity to correct it on the spot. If you do correct it and the program works, remember to save the corrected program to disk; you have altered only a copy of the program in

Reserved Words and Symbols in BASIC

" "	Characters inside quotation marks are processed as data, not executed as statements.
:	Separates statements or commands entered on a single program line.
;	Tells BASIC to execute the next statement without moving the cursor.
+ -	Signs for addition and subtraction. The plus sign also acts to link two character strings.
* /	Signs for multiplication and division.
$	Indicates that the content of a variable is to be treated as a non-numeric value.
=	Denotes equality in **if. . .then** statement; assigns new value to variable in **let** statement.
REM	Tells BASIC that remainder of program line is a remark, not to be executed.
RUN	Starts execution of the program BASIC holds in memory.
STOP	Causes program to stop execution. **Cont** causes program to resume.
LIST	Displays a line-by-line listing of the program BASIC holds in memory.
NEW	Clears BASIC program memory; used before entering a new program from keyboard.
LOAD"	Brings a specified BASIC program from a disk into program memory.
SAVE"	Copies the BASIC program currently in memory onto a disk; program must be named.

CLS	Clears the screen of all previous display. Does not affect the program in memory.
KEY	**Key off** turns off the function key indicators at bottom of screen; **key on** turns them on.
SCREEN	Sets BASIC for operation in one of its graphic or text modes.
LET	Assigns a value, or the product of a computation, to a variable.
INPUT	Assigns values to variables from the keyboard.
GOTO	Branches the program to a specified line.
GOSUB	Branches the program to a subroutine; **return** branches the program back.
IF. . .THEN	Causes a stated procedure to be executed if a specified expression is true.
FOR. . .NEXT	Causes the program to repeat a set of procedures for a stated number of times.
READ	Assigns to specified variables a set of values drawn sequentially from **data** statements.
PRINT	Displays specified expressions on the monitor.
LPRINT	Sends specified expressions to the computer's printer.
LOCATE	Positions the cursor at the specified line and column numbers.
END	Terminates program execution and returns BASIC to the immediate mode.

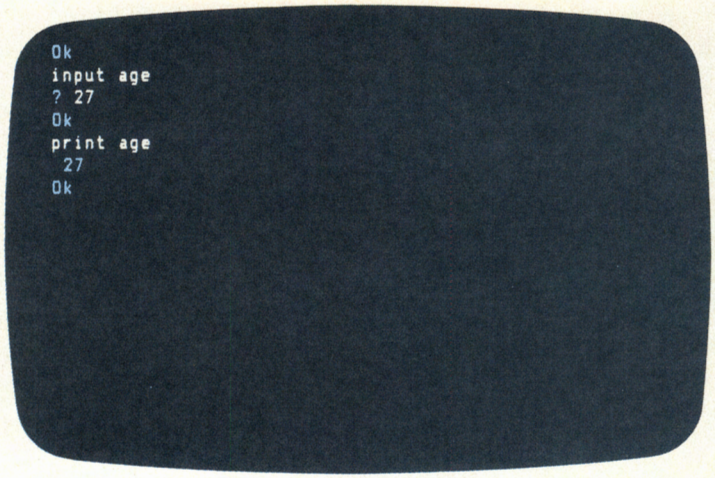
This chart lists words and symbols with special meanings in BASIC. Each tells BASIC to carry out a specific task; if the task is inappropriate or impossible, a "Syntax error" message will appear. If you misuse a key word, but the program line still makes sense, BASIC will execute the line — with varying results.

the computer's memory, not the one stored on the disk.

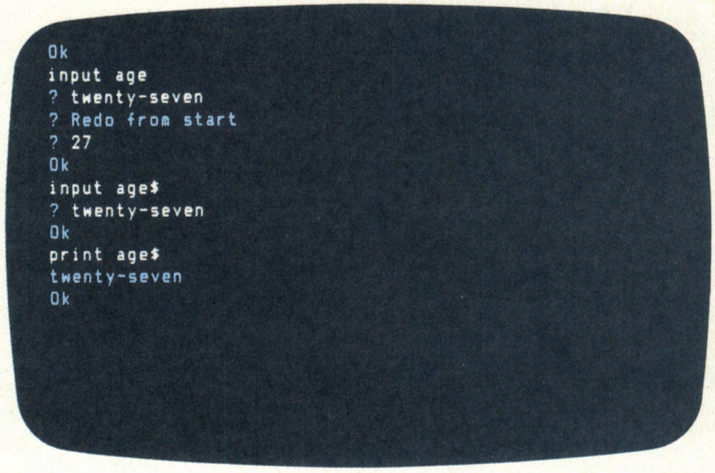
You can reduce the occurrence of error messages by checking each line before you enter it. If you find an error, you can correct it by moving the cursor back to the point of the error and typing over it; the newly typed material will replace the old. If you wish to insert material in the middle of a line, move the cursor to that point and press the **Insert** key; the cursor will change shape, and whatever you type will be inserted while the original material moves to the right. If you don't catch an error until after you have entered a line, you can fix the entire line by simply retyping it correctly, using the same line number. The BASIC interpreter will store the newly entered line in place of the line already stored in memory under that number.

One feature of BASIC that makes it easy for the novice to use is that key words are usually self-explanatory. Some instructions, called commands, act on an entire program — to store or retrieve it from memory, to list its contents or to execute it. Other instructions, called statements, work within the program, controlling processes that lead to the results the program is designed to produce. Many of the most common and useful commands and statements will be explained in this chapter.

Getting Data from the Keyboard



The *input* statement causes BASIC to read values from the keyboard. Here, BASIC stores the value it reads in the variable named *age*. When you enter the line, BASIC responds with a *?* prompt — indicating that you should enter a value from the keyboard. When you do so — in this case, typing the number *27* — BASIC assigns the number to the variable, *age*. The *print* statement instructs BASIC to display the specified information — the value of the variable, *age*.



A variable with a name like *age* can hold only numeric values. If you try to put anything else into such a variable, BASIC will tell you to start your input again. You can store letters in a variable if you have given the variable a name that ends with a *$*. This type of variable is called a string variable, and its content, which can be any combination of letters and numerals, is called a string.

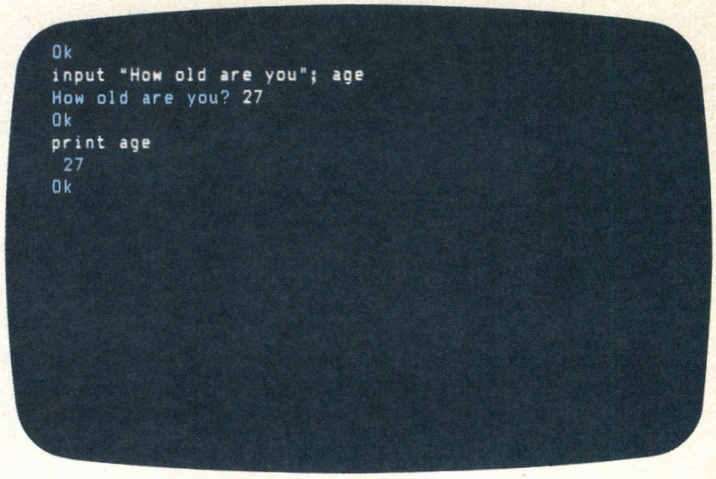
A variable is one of BASIC's most useful tools. BASIC uses variables as if they were file folders, each with its own name. You might store any value in the variable, but to manipulate that value, you need only instruct BASIC to perform an operation on the variable itself. This means that one program line can be used to do the same thing to any number of values, as long as each value is stored in its turn under the same variable name.

BASIC distinguishes between numeric variables, which can hold only numbers, and string variables, which can hold any combination of letters and numerals. BASIC cannot use numerals stored in a string variable for mathematical computations, since BASIC understands the content of a string variable only as a list of alphanumeric characters.

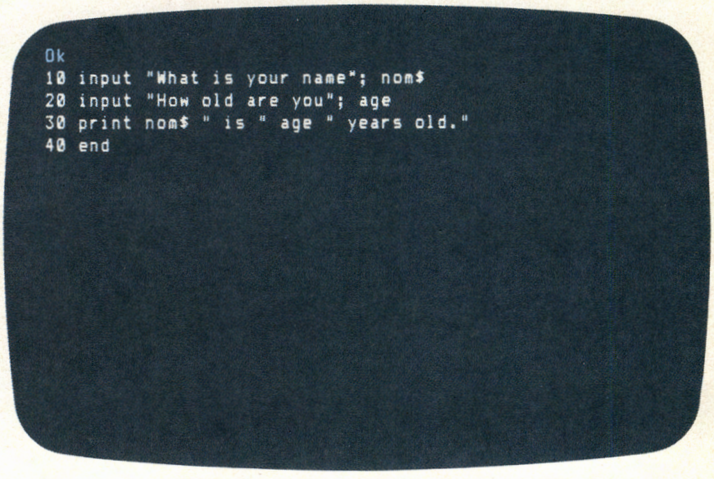
You can name a variable almost anything that reminds you what is stored there. Do not, however, use a reserved word that has special meaning for BASIC, or it will respond with a "Syntax error" message.

The most immediate way to assign values to variables is to use the *input* statement, which lets BASIC read keyboard data while a program is running. Keyboard input must be correctly structured; BASIC always expects the type and number of variables specified in the statement and will respond to any variation with "?Redo from start." If you get that message, retype your input correctly.

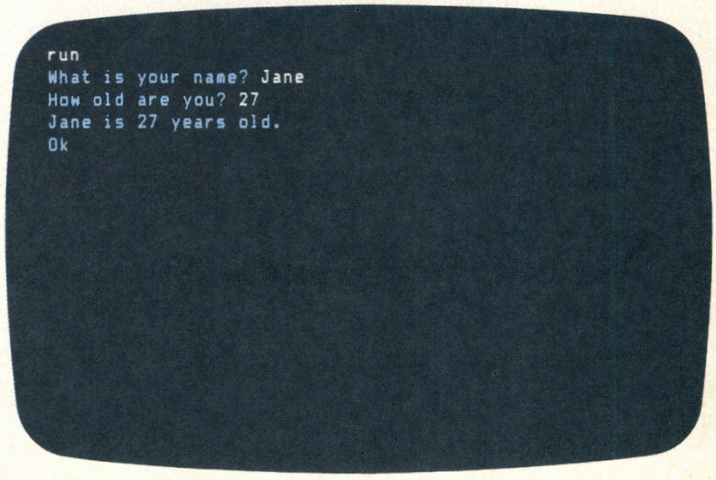
An *input* statement can contain a prompt that you determine, so that when the program is run, BASIC can ask the person at the keyboard for the specific information needed. When you use an *input* prompt, it must be enclosed in quotation marks and followed by a semicolon and the name of the variable. The quotes do not appear when BASIC prints the prompt. To suppress the question mark when BASIC prints the prompt, use a comma instead of a semicolon.



This short program incorporates *input* statements calling for first a string variable and then a numeric variable. The third program line tells BASIC to display a sentence using both of the variables. Quotation marks are used in the *print* statement to surround nonvariable character strings — in this case, words — that you want to display.



When you run the program, BASIC executes the lines in order, first prompting you to input your name; after you enter your name, it prompts you to input your age. As soon as that is entered, BASIC executes the next program line, displaying the sentence with the contents of the two variables filled in.



Putting Values into Variables

You can use a *let* statement to have BASIC assign a value directly to a variable. A *let* statement can assign a numeric value, a string value, or even the result of a computation. In this example, *x* and *y* are assigned numeric values; *n* is assigned a value equal to the sum of *x* and *y*. BASIC automatically computes the sum, which appears when you give the instruction **print n**.

```
Ok
let x=3
Ok
let y=7
Ok
let n=x + y
Ok
print n
 10
Ok
```

You can combine *input* and *let* statements to have BASIC perform calculations on numbers that you enter at the keyboard. In this example, the variable *myage* is defined by keyboard input; the variable *older* is defined by a *let* statement. Each time you change the value of *myage*, the value of *older* changes correspondingly.

```
Ok
input myage
? 27
Ok
let older = myage + 2
Ok
print older
 29
Ok
input myage
? 43
Ok
print older
 45
Ok
```

You can assign values to a variable from within a program by using a *let* statement. This allows you to take full advantage of the PCjr's computing power, since the value you store in the variable need not be simply a numeric or string value, but can instead be the result of a set of computations. Thus the value of a variable might be the result of computations involving other variables; the expressions within a *let* statement can be as complex as your program requires, provided they are in BASIC syntax.

The arithmetic operators used for mathematical computations in BASIC *(chart, page 79)* are similar to symbols used in algebra. But the BASIC equal sign (=) sometimes has a meaning quite unlike the algebraic one. In a *let* statement, this sign does not mean that the expressions on each side are identical; it means the value on the right is to be stored in the variable named on the left. Thus the statement **let n=n+1**, impossible in algebra, has meaning for BASIC. It tells BASIC to take the old value stored in the variable *n*, add *1*, then store the sum in the variable *n*.

Whenever you use numbers in a BASIC program, remember that BASIC does not understand the use of commas in numbers. If you type a number with a comma as input, BASIC will respond with "?Redo from start"; if you use a comma in a number within a *let* statement, BASIC will respond with "Syntax error."

The short program on the screen at right computes a minimum monthly installment payment, based on inputs of total cost and down payment. The *let* statement assigns a computed value to the variable *payment*. Notice the use of parentheses; BASIC performs the operations inside parentheses, then those outside. Also note the use of the single quotation mark in line 30; it indicates that the remainder of the line is a remark, and is not to be executed.

```
Ok
10 input "Cost"; cost
20 input "Downpayment"; down
30 let payment = (cost - down)/24   For 24 payments
40 print "Minimum payment is" payment
run
Cost? 3000
Downpayment? 1200
Minimum payment is 75
Ok
```

The *let* statement can also be used to assign a string value to a variable. You must enclose the string value in quotation marks; BASIC reads everything within the quotes, including spaces and punctuation marks, as part of the string. You can combine two strings by using the positive sign (+); BASIC executes this by appending the second string to the end of the first.

```
Ok
10 let a$ = "Hello, "
20 input "Name"; b$
30 print a$ + b$
run
Name? Jane
Hello, Jane
Ok
```

You can use a *let* statement to assign the current date or time to a variable. The expressions *date$* and *time$* call up values from the PCjr's clock, which you set when you booted the DOS disk *(page 33)*. These date and time values can be useful in many program applications to pinpoint the time when a particular procedure was performed.

```
10 let d$ = date$
20 let a$ = "Invoice dated "
30 let b$ = "Amount payable: "
40 input "Invoice amount"; c$
50 print a$ + d$
60 print b$ + c$
run
Invoice amount? $300
Invoice dated 03-26-84
Amount payable: $300
Ok
```

Making Decisions with If . . . then

BASIC can be instructed to perform an action when a specified expression is true, using the combination *if . . . then*. In line 30 of the program at right, BASIC is told to look at the value assigned to the variable *dozen;* if the value is greater than 200, then BASIC is instructed to print "Too heavy." Note that BASIC displays the message only if the specified expression is true.

```
10 input "Weight"; weight
20 let dozen = weight * 12
30 if dozen > 200 then print "Too heavy"
run
Weight? 8
Ok
run
Weight? 20
Too heavy
Ok
```

You can expand the *if . . . then* construction by using *else* to designate an alternative action if the specified expression is not true. Changing line 30 of the program as shown at left causes BASIC to print one message if the value of *dozen* is greater than 200, and another message if the value of *dozen* is not greater than 200.

```
10 input "Weight"; weight
20 let dozen = weight * 12
30 if dozen > 200 then print "Too heavy" else print "Weight OK"
run
Weight? 8
Weight Ok
Ok
```

Much of the work done by a program consists of comparing values and performing actions based on the results of each comparison. In BASIC, this instruction is the *if* statement, which usually includes the word *then*. The use of *if . . . then* in BASIC is similar to its use in English; for example, you might tell a child, "If the sky is blue, then go outside to play," or tell BASIC, "If *b* equals 2, then print *b*." But unlike a child, BASIC does nothing if the expression proves false; thus, if the sky is not blue, a child might still go outside, but if *b* does not equal 2, BASIC simply goes on to the next instruction.

Variations of the *if* statement make possible a range of comparisons and actions. You can give BASIC an alternative action to perform if the expression is untrue, by adding an *else* clause after the *if* statement *(above, bottom)*. If you use the words *and* and *or* in the *if* statement, BASIC will test two or more expressions, and base its decision on the combined result. An equal sign in an *if* statement, unlike one in a *let* statement, does mean "is equal to."

You can substitute *goto* for *then* in the *if* statement *(opposite, bottom left)*, directing BASIC to branch to a specified line number if an expression is true. *Goto* is useful in its own right; it can be used anywhere in the program without an *if* statement, to branch from the sequence of line numbers to any point in the program.

```
10 input "Temperature (centigrade)"; temp
20 input "Is it raining (y/n)"; y$
30 if temp < 0 and Y$="y" then print "Freezing rain"
run
Temperature (centigrade)? -3
Is it raining (y/n)? y
Freezing rain
Ok
run
Temperature (centigrade)? 4
Is it raining (y/n)? y
Ok
```

The *if* statement can be used to have BASIC make a decision based on the truth of more than one expression. In this program, if both of the expressions specified in line 30 are true, BASIC displays the message "Freezing rain." If either expression is not true, BASIC takes no action.

```
10 input "What is 2 + 2"; ans
20 if ans=4 goto 30 else goto 10
30 print "Correct"
run
What is 2 + 2? 3
What is 2 + 2? 5
What is 2 + 2? 4
Correct
Ok
```

By substituting *goto* for *then* in an *if* statement, you can cause a program to branch to a specified line number out of the ordinary sequence. In line 20 of this program, BASIC is directed to branch to line 30 if the value of *ans* is 4, and to branch to line 10 if the value of *ans* is not 4. The program returns to the beginning each time the input differs from what you have directed BASIC to look for. When the expected input appears, the program displays the message "Correct."

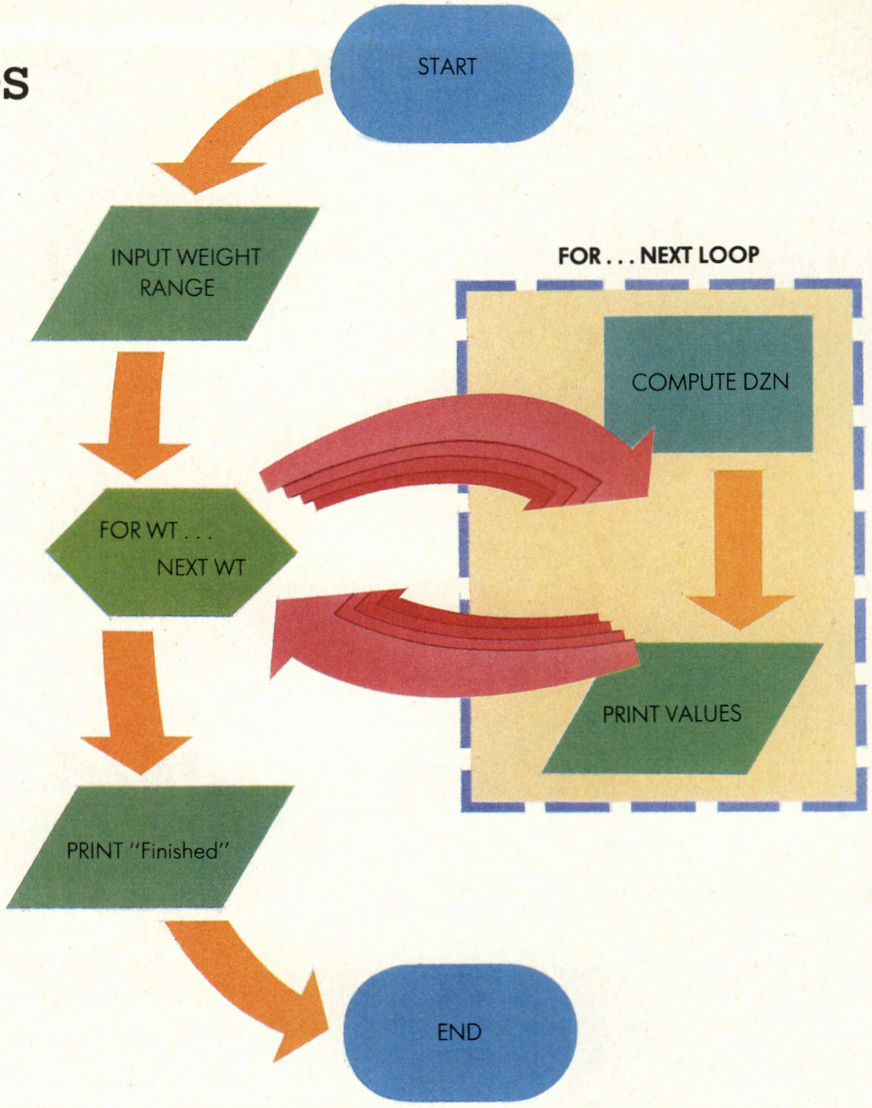
To have BASIC test more than one expression and perform an action if any one of the expressions is true, use *or* in the *if . . . then* construction. If either expression in line 30 is true, BASIC displays the message "Not here, please." Only one expression in an *if . . . or . . . then* statement need be true for BASIC to perform the designated action, and three or even more expressions can be examined in one *if* statement.

```
10 input "Do you smoke (y/n)"; ans1$
20 input "Does anyone in your office smoke (y/n)"; ans2$
30 if ans1$="y" or ans2$="y" then print "Not here, please."
run
Do you smoke (y/n)? n
Does anyone in your office smoke (y/n)? n
Ok
run
Do you smoke (y/n)? n
Does anyone in your office smoke (y/n)? y
Not here, please.
Ok
```

You can use *if* statements in successive lines to sift through a number of possibilities; if BASIC finds the expression in the first statement to be false, it ignores the rest of the line, goes on to the next line and performs the next test. This program is set up to return to the start unless the input response is *n* or *N*.

```
10 input "Do you want to go in circles";ans$
20 if ans$="y" or ans$="Y" then print"Here we go":goto 10
30 if ans$="n" or ans$="N" then print"Goodbye" else goto 10
40 end
run
Do you want to go in circles? y
Here we go
Do you want to go in circles? x
Do you want to go in circles? 37
Do you want to go in circles? n
Goodbye
Ok
```

A Loop to Repeat a Set of Procedures



When you develop any but the simplest program, it is useful to outline the procedure you have in mind before you try to write the BASIC source code of the program. This flow chart outlines the program on the screen at the bottom of the opposite page, breaking it into each of its steps from beginning to end, including the *for . . . next* loop inside the broken line.

Note that the shapes used in flow charts in this book are similar to standard shapes used by most programmers. While your flow charts may not be as elaborate as this one, you should use the same shapes, for clarity when you read them later.

A computer excels at jobs that call for repeating the same process many times. You can make use of this power in BASIC by bracketing between two special statements — *for* and *next* — a procedure that you need to repeat; such bracketing causes BASIC to execute a loop, going through the same procedure as many times as you specify.

A *for* statement tells BASIC that it is to do a job a given number of times, as specified by the values assigned to a variable in the statement. The variable in a *for* statement is used as a counter; BASIC stores the initial value of the variable in memory, then executes the subsequent instructions until it reaches a *next* statement. BASIC then branches back to the *for* statement, adds the specified amount to the counter variable, and checks to see whether the value exceeds the specified final value. BASIC repeats the loop until the value of the counter exceeds the final value; then it branches directly to the statement following the *next* statement.

The power of the *for . . . next* loop lies in the way it can process a different set of values at each repetition. A good example of its usefulness is in chart preparation *(opposite, center and bottom)*. The loop might contain complex instructions for calculating and displaying data, but you would need to write the instructions only once, no matter how many lines were required to complete the chart.

The *for* statement in line 10 of this program tells BASIC to assign an initial value of 1 to the variable *greet,* then to execute all subsequent statements until it reaches a *next* statement. The *next* statement tells BASIC to increase the value of *greet* by 1 and see whether that value exceeds 5, the limit set by the *for* statement. If it does not, BASIC goes through the loop again; when the value of *greet* exceeds the limit, the program continues with the line following the *next* statement.

```
10 for greet=1 to 5
20 print "Hello there"
30 next greet
40 print "Goodbye"
50 end
run
Hello there
Hello there
Hello there
Hello there
Hello there
Goodbye
Ok
```

You can change the variable in a *for . . . next* loop by any amount, using a *step* expression, as shown in line 10. Note that when the program is run, it uses values for the variable *wt* that increase by 2 each time, but only up to 20. This is because the next *step* would increase the value of *wt* to 22, exceeding the limit of 21, so the program branches to the line following the *next* statement.

```
10 for wt=10 to 21 step 2
20 let dzn=wt*12
30 print "Weight =" wt, "Weight per dozen =" dzn
40 next wt
50 print "Finished"
60 end
run
Weight = 10    Weight per dozen = 120
Weight = 12    Weight per dozen = 144
Weight = 14    Weight per dozen = 168
Weight = 16    Weight per dozen = 192
Weight = 18    Weight per dozen = 216
Weight = 20    Weight per dozen = 240
Finished
Ok
```

You can set the limits of a *for . . . next* loop using values assigned to variables with an *input* statement, as shown at right. Each time the program runs, the starting and ending point of the loop are set from the keyboard. Note the structure of the *input* statement and of the keyboard input; a comma must separate the variable names and the keyboard entries when you assign values to more than one variable.

```
10 input "Weight range (from, to)"; w1, w2
20 for wt = w1 to w2
30 let dzn=wt*12
40 print "Weight =" wt, "Weight per dozen =" dzn
50 next wt
60 print "Finished"
70 end
run
Weight range (from, to)? 6, 9
Weight = 6     Weight per dozen = 72
Weight = 7     Weight per dozen = 84
Weight = 8     Weight per dozen = 96
Weight = 9     Weight per dozen = 108
Finished
Ok
```

Storing and Retrieving Data

A *read* statement instructs BASIC to establish one or more variables, then to look for the first *data* statement in the program and assign values it finds there to the variables in the *read* statement. In the short program at right, the *read* statement assigns values to three variables, which are then displayed by the *print* statement. Note the use of commas in the *print* statement; each tells BASIC to display the next listed element in the next screen column. These invisible columns are always 14 spaces wide.

```
10 read x,y,z
20 print x,y,z
30 end
40 data 2,4,13
run
 2             4             13
Ok
```

When a *read* statement looks at a *data* statement more than once, it moves to the next set of data that it has not already used. In the program shown at right, the *read* statement is placed in a loop that causes it to be repeated five times; at each repetition, it reads a new set of data, which is displayed by the *print* statement. A *read* statement can assign values to numeric or string variables; string values — in this case, names — do not need to be bracketed with quotation marks in the *data* statement unless they contain symbols that have a special meaning in BASIC *(page 79)*.

```
10 for n=1 to 5
20 read place, runner$
30 print place, runner$
40 next n
50 end
60 data 1,Jane,2,John,3,Julia,4,Jack,5,Jennie
run
 1            Jane
 2            John
 3            Julia
 4            Jack
 5            Jennie
Ok
```

Many programs process information that does not change from one run to the next. It is useful to keep this data permanently attached to the program. You could assign the data values to variables using *let* statements, but this would be a cumbersome procedure if you had more than a few items. BASIC provides a much easier way to deal with this situation; it employs a pair of statements called *read* and *data*.

A *read* statement tells BASIC to establish one or more variables, and then to fill them with values taken sequentially from a *data* statement. The first *read* statement in a program draws on the first *data* statement, wherever it may appear. BASIC takes as many values from the *data* statement as are required to fill the variables of the *read* statement. BASIC keeps track of the values that have already been used, by leaving a pointer in the *data* statement; the next time a *read* statement returns to

You can use *read* and *data* statements to supply values for repeated calculations, as shown in this program. The *read* statement assigns values to variables that are then used in line 40 to compute other values; the results are displayed by the *print* statement. The zeroes at the end of the last *data* statement are end-of-data markers; when these values are assigned to variables, the *if* statement in line 30 directs BASIC to branch to the *end* statement.

```
10 print "Year", "Income", "Expenses","Net","Percent"
20 read year$, inc, spent
30 if year$="0" goto 70
40 let net=inc-spent: let pct=100*(net/inc)
50 print year$, inc, spent, net, pct
60 goto 20
70 end
80 data 1974,32578,31432,1975,29864,30123,1976,31589,30215
90 data 1977,34769,33350,1978,36801,34982,1979,36544,34700
100 data 1980,37297,35263,1981,38477,35802,1982,39469,36223
110 data 1983,41005,37456,0,0,0
```

When the program shown in the screen above is run, all of the values from the *data* statement are displayed in a chart, along with some values computed using the data. You could add new figures for each succeeding year simply by inserting new values into the *data* statements, before the marker zeroes that indicate the end of the data.

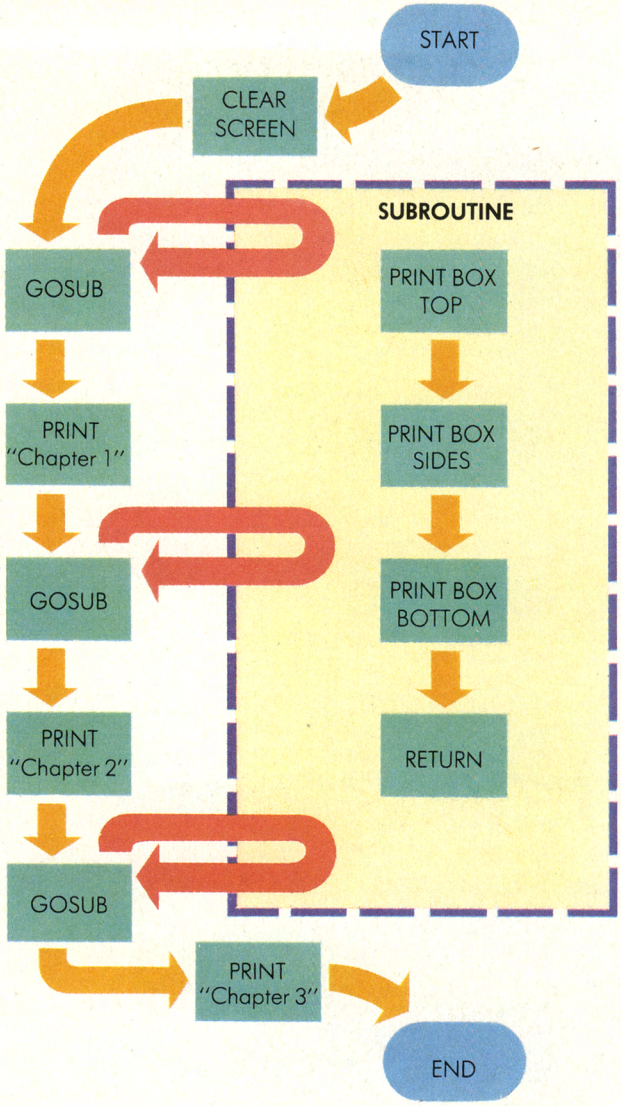
```
Year          Income        Expenses      Net           Percent
 1974          32578         31432         1146          3.517711
 1975          29864         30123         -259         -.8672649
 1976          31589         30215         1374          4.349616
 1977          34769         33350         1419          4.081222
 1978          36801         34982         1819          4.942801
 1979          36544         34700         1844          5.045972
 1980          37297         35263         2034          5.453522
 1981          38477         35802         2675          6.952205
 1982          39469         36223         3246          8.224176
 1983          41005         37456         3549          8.655042
Ok
```

the *data* statement, it starts with the first unused values.

When BASIC comes to the end of a *data* statement and still needs values to fill the variables of a *read* statement, it moves to the next *data* statement and continues to read. If it runs out of data before all of the variables are filled, BASIC stops running the program and displays an error message. You can avoid this problem by putting a special marker value — a zero where BASIC expects to find a year, for example — at the end of your data, and writing an *if* statement that tells BASIC what to do when the *read* statement assigns this value to a variable *(line 30, top screen, above)*. Another way to avoid an end-of-data error is to write a special error-trapping routine; this more advanced procedure is explained in the IBM *BASIC* manual.

You can reuse the values in a *data* statement by resetting the pointer. A *restore* statement does this job, telling BASIC to move the pointer back to the beginning of the first *data* statement *(see the program listed on page 94)*. You can also use a *restore* statement anywhere in the program to set the pointer to the beginning of a specific *data* statement: Simply follow the *restore* statement with the line number of the *data* statement you want to start with.

Subroutines to Be Called When Needed



This flow chart illustrates the working of the program shown at the bottom of the opposite page. The subroutine for printing a box around a title is like a shorter program, and it performs the same job each time it is used. BASIC branches to the subroutine when it encounters a *gosub* statement; when it reaches a *return* statement, it resumes execution of the main program, with the statement that follows the *gosub*.

Many programming situations require that you repeat a particular set of steps at several different points within the program. Instead of writing the same set of instructions several times, you can write the code once, in a sort of miniature program called a subroutine, and then tell BASIC to branch to the subroutine each time you need it.

This procedure, which is known as calling a subroutine, is made possible by the *gosub* and *return* statements. A *gosub* tells BASIC to branch to the subroutine that begins at a specified line, then execute all subsequent lines until it encounters a *return* statement. The *return* instructs BASIC to branch back to the statement following the *gosub* that called the subroutine. A subroutine can be called any number of times, and from anywhere in the program.

All subroutines should be positioned after the *end* statement of a program, to prevent BASIC from executing the subroutine as it proceeds sequentially through the numbered lines of the program. If BASIC encounters a *return* statement that does not have a preceding *gosub*, it will stop the program and display an error message.

```
10 print "Hello."
20 gosub 1000
30 print "This program demonstrates subroutines."
40 gosub 1000
50 print "It goes to the 'Pause' subroutine at"
60 print "line 1000 after each message."
70 gosub 1000
80 print "This ends the demonstration."
90 end
1000 rem 'Pause' subroutine
1010 for n=1 TO 3000:next n
1020 print
1030 return
```

When BASIC reaches a *gosub* statement anywhere in this program, it branches to the subroutine at line 1000. The subroutine is a simple *for . . . next* loop that counts to 3000, followed by a *return* statement that causes BASIC to branch back to the statement following the most recent *gosub*.

When you run the program shown at left, BASIC displays the sentences in the *print* statements *(below)*. Each sentence is followed by a blank line and a pause of about two seconds, both inserted by the subroutine. A *for . . . next* loop is often used to create a pause in BASIC. To vary the pause length, change the final value of the counter in the *for* statement; it takes about a second for your PCjr to count to 1250 in this type of loop.

```
Hello.

This program demonstrates subroutines.

It goes to the 'Pause' subroutine at
line 1000 after each message.

This ends the demonstration.
Ok
```

```
10 'Prints headings in boxes
15 cls
20 gosub 2000
30 locate 2,3:print "Chapter 1"
40 locate 5,1:gosub 2000
50 locate 6,3:print "Chapter 2"
60 locate 9,1:gosub 2000
70 locate 10,3:print "Chapter 3"
80 locate 13,1
1000 end
2000 'Subroutine to print box
2010 let a$=chr$(218):b$=chr$(196):c$=chr$(191)
2020 let d$=chr$(179):e$=chr$(192):f$=chr$(217)
2030 print a$;:for n=0 to 12:print b$;:next n:print c$
2040 print d$;:for n=0 to 12:print" ";:next n:print d$
2050 print e$;:for n=0 to 12:print b$;:next n:print f$
2060 return
```

The program above uses a subroutine and the graphics characters of your PCjr to create boxes around text on the monitor screen. The values assigned to variables in lines 2010 and 2020 of the subroutine are ASCII codes for the graphics characters that will make up the boxes; these codes are listed in an appendix to the *BASIC* manual. Note the use of line numbering and remarks to make the program easy to read.

When the program at left is run *(below)*, each box is drawn, then the text is printed in it. *Locate* statements position the text by telling BASIC where to put the cursor before each *print* statement. The numbers in a *locate* statement indicate first the line, from 1 to 25, and then the column, from 1 to 80.

```
┌─────────┐
│Chapter 1│

┌─────────┐
│Chapter 2│

┌─────────┐
│Chapter 3│

Ok
```

Sound and Graphics: a Treasury of Effects

A *sound* statement sets two values that control the frequency and duration of a sound generated by the PCjr. The values may be numbers or variables, as shown in this program. In line 50, frequency is set by the variable *freq,* which is defined in line 40; duration remains constant — for each note, 20 ticks of the PCjr's clock *(text, below).* In line 100, frequency remains constant (200 cycles per second) and duration is set by the variable *dur (line 80).* A second *sound* statement in line 100 causes a pause; BASIC executes frequency 32767 as silence.

```
10 'Demonstrates SOUND statement
20 play "mf"
30 locate 13,15:print"Frequency and duration in the SOUND statement"
40 for freq=100 to 400 step 20
50 sound freq, 20
60 locate 15,20:print"This is frequency" freq ", duration 20"
70 next freq
80 for dur=10 to 60 step 10
90 locate 16,20:print"This is frequency 200 , duration" dur
100 sound 200, dur: sound 32767,10
110 next dur
run
              Frequency and duration in the SOUND statement

                   This is frequency 400 , duration 20
                   This is frequency 200 , duration 30
```

This program uses two *play* statements to play the long sequence of notes and pauses of a musical piece. Lines 20 through 90 assign to variables the character strings that represent actual musical notation. In line 120, the first element of the *play* statement sets the tempo of the music at 180 quarter notes per minute. The remaining elements in the *play* statements tell BASIC to execute each string named after the letter *x.* Each of these elements must be followed by a semicolon, or BASIC will stop the program and signal "Syntax error."

```
10 'Program to play something like a waltz
11 '
19 'Assign "subtunes" to string variables
20 A$="o3c8.p16ceg8.p16gp4g8p8g8p4"
30 B$="e8p8e8p4": C$="f8p8f8p4"
40 D$="o2b8.p16bo3da8.p16ap4a8p8a8p4"
50 E$="f8p8f8p4": F$="e8p8e8p4"
60 G$="c8.p16cego4cp4c8p8c8p4"
70 H$="o3g8p8g8p4": I$="o3a8p8a8p4"
80 J$="o3d8.p16dfa8.p16a2g-g8..o4e2p8co3ee.da.gc."
90 K$="p8msc16c8p32c."
100 '
110 'Play waltz as series of subtunes
120 PLAY"t180;xa$;xb$;xa$;xc$;xd$;xe$;xd$;xf$;
130 PLAY"xg$;xh$;xg$;xi$;xj$;xk$;"
run
```

You can use BASIC to harness the considerable sound and graphics capabilities of the PCjr for programs you create. Under the direction of Cartridge BASIC, the PCjr can produce up to three different sounds simultaneously, using its three "voices." Each voice can be programmed with a *sound* or *play* statement. A *sound* statement *(above, top)* lets you define the pitch and length of a sound by specifying a frequency, in cycles per second, and a duration, expressed as a number of ticks of the PCjr's internal clock — which ticks 18.2 times per second. You also have a choice of 16 volume settings.

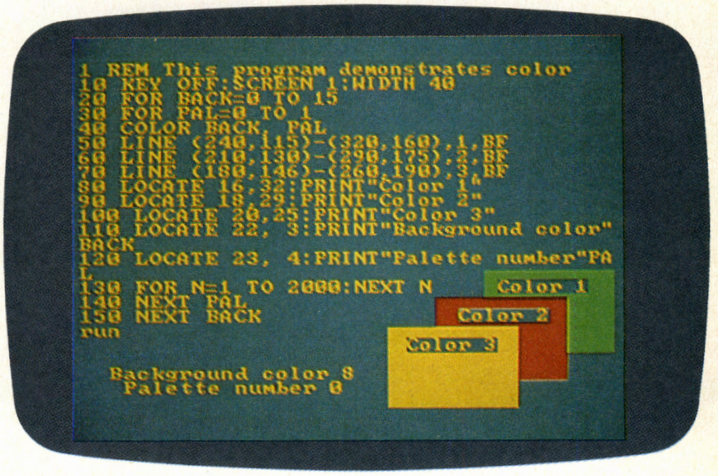
Each *sound* statement defines only one tone; to write and play music, you can use *play* statements. Like a *print* statement, a *play* statement processes strings of characters within quotation marks. But the strings in a *play* statement have meaning only as musical notation; instead of displaying the characters on the monitor, BASIC interprets them as instructions for the speaker. The strings in a *play* statement must be written according to a strict set of rules, explained in IBM's *BASIC* manual.

The PCjr is every bit as versatile in its visual performance as it is with sound. BASIC uses *screen* statements to control the display of material on a monitor. *Screen 0* sets a mode that allows the display of only text and IBM's special character set. Within this mode you can use a *width 40* or a

When BASIC is in text mode, as set by a *screen 0* statement, a *color* statement controls the background color of the screen and the foreground color of the displayed characters. Here, the *input* statement on line 30 assigns values to the variables *FORE* and *BACK;* the *color* statement on line 40 uses these values to set the colors of the message displayed by the *print* statement on line 60. The *if . . . then* statement on line 50 tells BASIC to end the program if the normal color setting of *7,0* is selected. A text *color* statement accepts numbers in the range 0 to 31 for the foreground color, and in the range 0 to 7 for the background.



When BASIC is in the medium-resolution graphics mode, as set by a *screen 1* statement, use the *color* statement to select one of 16 background colors and one of the two "palettes" of three colors that can be displayed by the other graphics statements. In the program at right, two *for . . . next* loops change the variables in the color statement on line 40, causing BASIC to set each palette and background color in turn. The *line* statements on lines 50 through 70 set coordinates for the corners of three rectangles, each filled with one of the three colors of the current palette.



width 80 statement to set the number of columns (40 or 80). When you use a *color* statement in text mode, you can draw from 16 colors for the characters, the background and the screen border.

Two graphics modes allow you to use BASIC's sophisticated graphics statements. The medium-resolution graphics mode, set by a *screen 1* statement, also lets you use any four of the 16 available colors in each screen you create. *Screen 2* sets the high-resolution graphics mode. In this mode you can use all of the graphics statements, such as *line*, *draw* and *circle*, but not the *color* statement; all shapes, lines and text appear as white on black. Text is displayed in 40-column width in the medium-resolution mode; in the high-resolution mode, text is in 80-column width. Four more *screen* statements, available only in Cartridge BASIC, provide a greater color choice in the graphics modes.

There are many subtleties to the use of the sound, color and graphics capabilities of your PCjr, only hinted at by the examples above. If you use the explanations in the IBM *BASIC* manual as guides, you will find that BASIC can put a new world of sight and sound at your fingertips.

Writing Your Own Programs

```
10 CLS:KEY OFF:SCREEN 1:WIDTH 40:COLOR 9,0
20 LOCATE 5,15:PRINT"WELCOME"
30 LOCATE 9,9:PRINT"to the Short Report"
40 LET T=3000:GOSUB 2000
50 SCREEN 0,1:WIDTH 80:COLOR 0,7:CLS
60 GOSUB 4000
70 GOSUB 1000
80 COLOR 7,0:CLS:LOCATE 5,10:PRINT"Program ends after 10 entries."
99 END
1000 '       Subroutine for input of year desired
1005 FOR ENTRY=1 TO 10:CLS:COLOR 0,7:LOCATE 5,10
1010 INPUT "What year for your report (or 0 to stop)";YEAR$
1020 IF YEAR$="0" THEN END ELSE GOSUB 3000
1025 LOCATE 20,10:COLOR 5,7
1030 INPUT"Press RETURN to continue...",A$
1035 NEXT ENTRY
1040 RETURN
2000 '       Subroutine for timer
2010 FOR COUNT=0 TO T:NEXT COUNT:RETURN
3000 '       Subroutine to calculate and display values
3010 RESTORE
3020 READ Y1$,INC,SPENT:IF Y1$="0" THEN GOSUB 5000:RETURN
3030 IF Y1$<>YEAR$ GOTO 3020
3040 LET NET=INC-SPENT: LET PCT=100*(NET/INC)
3045 CLS:LOCATE 5,3
3050 PRINT "Report of ";DATE$;" for user";CODE;"at ";TIME$
3060 LOCATE 8,3:PRINT "Year","Income","Expenses","Net $","Percent"
3070 IF NET<0 THEN COLOR 4,7 ELSE COLOR 1,7
3080 LOCATE 10,3:PRINT Y1$, INC, SPENT, NET, PCT
3090 COLOR 0,7:RETURN
4000 '       Subroutine to establish user identity
4010 LOCATE 5,10:INPUT "Please enter your secret code: ",CODE
4020 IF CODE>127 OR CODE<125 THEN PRINT"Sorry, you are not authorized.":END
4030 RETURN
5000 '       Subroutine for no data
5010 COLOR 7,0:LOCATE 7,10:PRINT " No data for year ";YEAR$;" "
5020 LET T=1500:GOSUB 2000:RETURN
7000 DATA 1974,32578,31432,1975,29864,30123,1976,31589,30215
7010 DATA 1977,34769,33350,1978,36801,34982,1979,36544,34800
7020 DATA 1980,37297,35263,1981,38477,35802, 1982,39469,36223
7030 DATA 1983 ,41005,37456
9999 DATA 0,0,0       :'This marker stops the search.
```

The program listed at right permits users who have typed in a secret code number to read financial data for selected years. The body of the program *(orange)* sets up screen widths and colors, and directs the overall flow of the program. The subroutines *(yellow)*, which follow the *end* statement, do the work of taking input, reading data, computing information and displaying messages. The data the program uses *(green)* follows the subroutines. Note the *restore* statement in line 3010, which sets the data pointer *(page 89)* to the beginning of the first *data* statement each time the subroutine uses the data. To use this program, you must first enter a three-digit code number, controlled by the *if* statement on line 4020.

The best way to write a BASIC program is one piece at a time. First define the jobs that need to be done within the program, then write the subroutines, loops and lists of statements to do each of the jobs. This way you can test and correct each part of the program independently before all the pieces are together; you will have far less trouble finding the bugs in a short segment than in an entire program that may be hundreds of lines long.

Building a program in segments also helps you keep a logical program flow, which is easier to understand. And if each part is largely self-contained, you can more easily modify a part of the program without affecting the rest.

With the elements of BASIC you have learned in this chapter, you can begin writing programs on your own. Keep in mind, however, that you have only scratched the surface. There are more than 100 commands and statements in the syntax of Cartridge BASIC, each designed to do a specialized job. It takes time to become familiar with the language and all that it can do for you. You can learn a great deal by reading other people's programs, trying to follow the flow of control and determine the outcome of the processes. However, the best way to master BASIC is by writing programs of your own, trying out new ways of doing things. Practice makes programmers.

The first screen displayed by the program is controlled by the *screen*, *width*, *color*, *locate* and *print* statements in lines 10 through 30. It remains on the screen while the program branches from line 40 to the timer subroutine at line 2000. BASIC then returns to line 50, where new *screen*, *width* and *color* statements establish the initial settings for the next subroutine. Line 60 branches the program to the user-identity subroutine at line 4000; when that is completed, BASIC returns to line 70 and immediately branches to the input subroutine at line 1000.

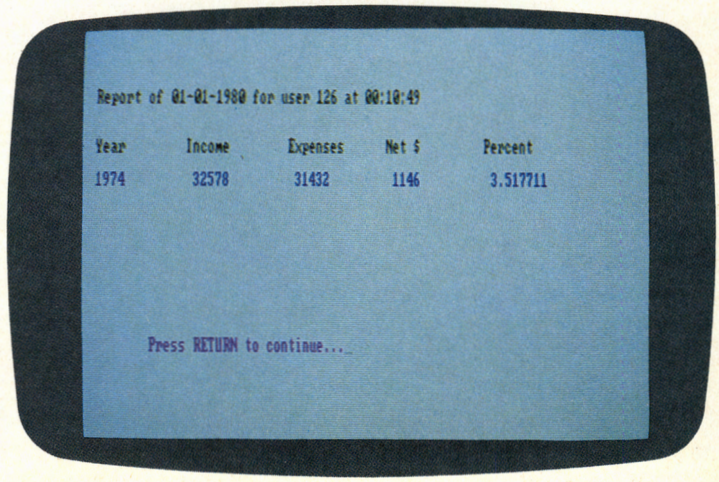


The input subroutine first displays the message shown at the top of this screen. If your selection is *0*, the program ends. Otherwise BASIC branches to the subroutine at line 3000, which reads, calculates and displays data. If there is no data for the selected year, BASIC branches to the subroutine at line 5000, which prints the "no data" message. When BASIC returns to the point where it left the input subroutine, the "continue" message is displayed; when you press the **Enter** key, BASIC goes back to the beginning of the subroutine for another pass through the *for . . . next* loop.



If the program has data for the year you select, the subroutine at line 3000 puts together a report similar to the one shown at right. Note that the date and time, inserted by line 3050, are correct only if you entered the correct date and time when you booted your PCjr *(page 33)*.

The input subroutine repeats itself 10 times, allowing you 10 entries before program control returns to line 80, which prints a final message before the program ends.



Appendix

DOS Error Messages

Errors are a hazard in almost any process; working with computers is no exception. Sometimes the problem is with the hardware, sometimes it lies in the software — and very often it is caused by a misunderstanding between you and the equipment, soft or hard. For example, if you unknowingly slip out of your word processing program and into DOS and then type a command to the word processor, DOS will respond, **Bad command or filename**. Thus, the first thing to do is determine whether an error message is coming from the application program, from BASIC or from DOS.

This chart covers a few of the error messages most frequently issued by DOS. If you are in BASIC, you should refer to the IBM *BASIC* manual; if you are using an application program, you will have to refer to its own documentation.

ATTEMPTED WRITE-PROTECT VIOLATION
signals that the disk you are trying to format has its write-protect notch covered. Do not remove the tab until you have thought about why you put it there in the first place; you may destroy important data.

BAD COMMAND OR FILENAME
indicates one of two things: either that the command you have entered is not a valid DOS command or that DOS could not find the file you named, when it searched the disk in the drive. (**File not found** is a related message.) Check your spelling, of both the command and the filename; also check your disk directory.

COMPARE ERROR(S) ON TRACK XX, SIDE XX
occurs when you use the command **diskcomp** to try to compare two disks that are not identical. If you copy files from one disk to another using the **copy** command, all the information is transferred, but it is not stored in the same tracks on the target disk as it was on the source disk. This causes discrepancies when the disks are compared track for track. If you want to make an exact copy of a disk, use the **diskcopy** command.

DATA ERROR READING DRIVE A:
ABORT, RETRY, IGNORE?
often means a disk has a defective spot. Enter **r** for retry. If this does not correct the error, abort the program and remove the disk. Note: The *ignore* response is not recommended, because it will destroy data. If you decide to reformat the disk, be sure to copy files first.

DISK BOOT FAILURE
may mean the disk drive door is open, or that the disk is wrong side up. If neither is the case, copy the files from the failed disk to a new one and boot from the new one.

FORMAT FAILURE
and related messages such as **Unable to write boot** or **Unrecoverable format error on target** may arise with the **format**, **diskcopy** or **system** commands. If the problem is the target disk, discard it and use another; if it is the source disk, copy all files and then reformat the disk. (Note: Consistent problems with disks you have formatted may indicate a disk-drive hardware problem and should be reported to your service center.)

INSERT COMMAND.COM DISK IN DRIVE A:
AND STRIKE ANY KEY WHEN READY
may occur when DOS is trying to reload the command processor but cannot find it on the disk in the drive. Insert a DOS disk and press any key. A related message — **Invalid command.com in drive a:** — means that the *command.com* on the disk is not the right version.

NON-DOS DISK
signals that the file allocation table does not contain valid information. Run *chkdsk* to see if the problem can be corrected. If not, the disk probably needs to be reformatted, but try to copy important files first.

NON-SYSTEM DISK OR DISK ERROR
REPLACE AND STRIKE ANY KEY WHEN READY
means there is no entry for *ibmbio.com* or *ibmdos.com* in the directory, or that the disk drive had trouble with the disk when you booted up. Insert a DOS disk and reboot.

Taking Care of Your System

ENVIRONMENTAL PROTECTION

Keep all food and liquids away from the computer, and do not smoke in the same room with the machine.

Place the computer at least three feet away from vents, air conditioners and radiators, and six feet away from space heaters.

Do not place the computer in direct sunlight; the heat could overwhelm the machine's cooling system.

Keep the computer from three to six feet away from telephones, tape recorders and other devices that can generate magnetic fields; the fields can interfere with the disk drives, memory circuits and disks. If necessary, shield the computer from these devices with a metal bookcase or file cabinet. For the same reason, keep objects that can become magnetized, such as car keys and paper clips, at a safe distance from the machine and the disks.

To keep static electricity to a minimum, try to maintain humidity at 50 per cent or higher; use antistatic sprays and floor mats.

Avoid using auxiliary heating devices — especially kerosene heaters and wood stoves — in the same room with the computer. These heaters generate pollutants that can leave a film of kerosene, ash and dust on and in the machine.

Clean the system components and work area weekly. Use a vacuum cleaner, or wipe down the exterior of the machine with a soft, damp cloth. You may also use special products made for cleaning electronic equipment: lint-free cloths, pressurized air cannisters, screen-cleaning solutions.

Do not try to clean inside the system unit or the disk drives; close disk-drive doors to keep out dust.

Never use detergents, chemical solvents or all-purpose sprays to clean the components. These leave damaging residues, which can cloud the screen surface and remove its antiglare coating.

POWER PROTECTION

Connect the computer to a separate circuit if possible, and use a surge protector to guard against voltage spikes. Avoid outlets on the same circuit with large appliances such as refrigerators.

Make sure every component of the system is properly grounded with three-pronged plugs. Do not cut or bend the third prong, or use two-prong adapters; this defeats the purpose of the ground.

Keep all cables out of the way so they will not be stepped on, tripped over or pulled loose during a work session.

DISK PROTECTION

If possible, make backup copies of program disks before using them for the first time, and then work with the backups instead of the masters. Also back up important data disks frequently. Label and date the backup disk, and store it in a different place from the original. Metal file cabinets make good storage places for disks.

Always use a felt-tip pen to write on disk labels; pencils or ballpoints will damage the disk's surface and interfere with information stored there.

Store disks vertically or horizontally, but do not stack them more than 10 deep and do not bend them or crowd them into a container.

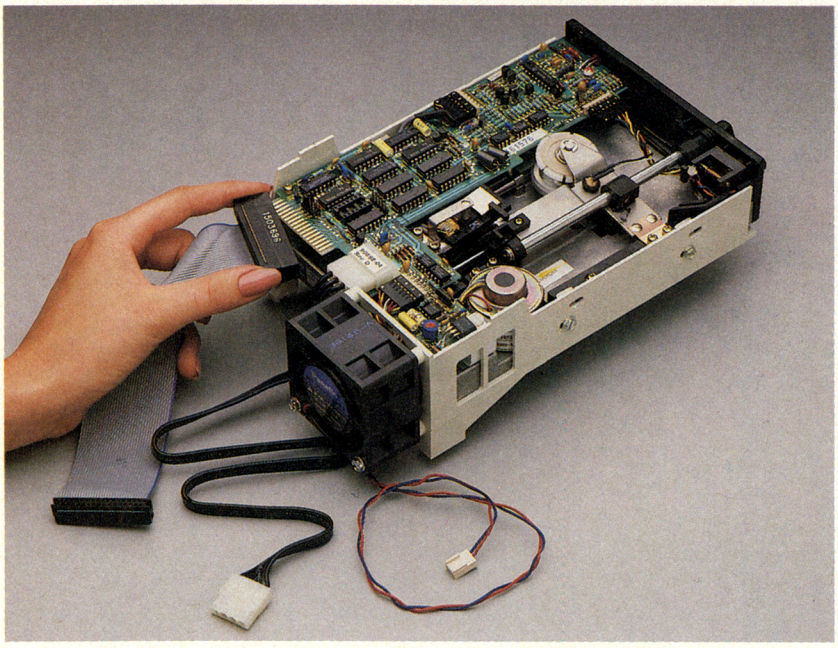
Never touch any exposed surface of the disk; skin oils will damage the surface, and any attempt at cleaning will only cause more damage.

Periodically inspect the center hole of the disk; if it is damaged, the disk should be replaced. Also inspect the surface area that is visible through the head slot. Shiny rings, scratches or folds are signs the disk should be replaced.

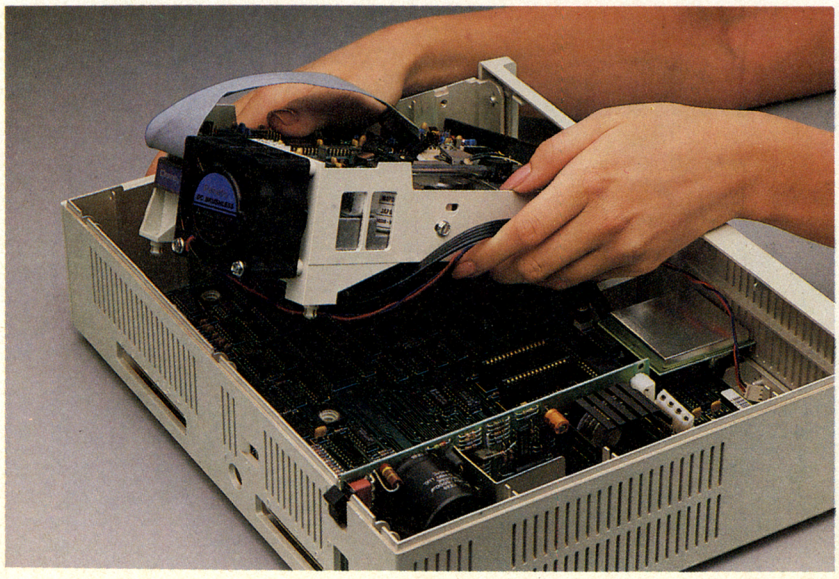
Use corrugated cardboard to mail floppy disks; regular cardboard is not stiff enough. Special mailers are also available from computer supply firms.

Upgrading an Entry-level System Unit

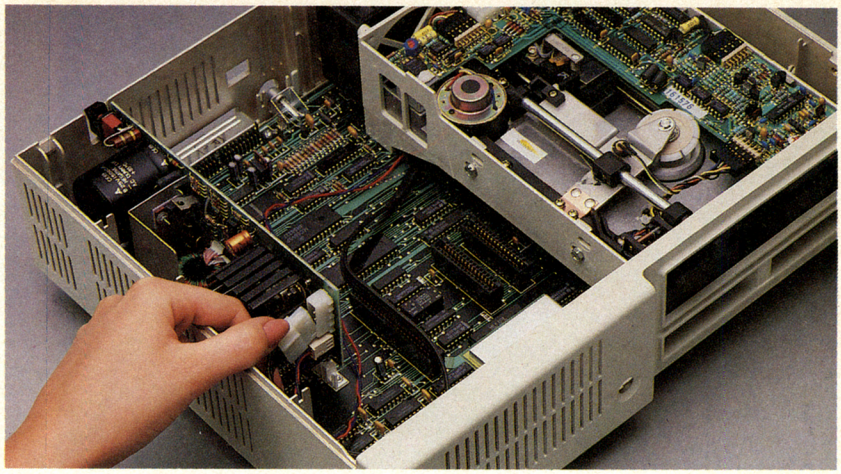
The first device you will want to add to your entry-model PCjr is an IBM Disk Drive. Before you make any connections, turn the computer's power switch off, disconnect the power cable and let the computer cool for five minutes. Then remove the cover from the system unit *(page 68)*. Unpack the disk drive unit, its two unattached cables, and the disk-drive adapter board. Plug one end of the black disk-drive power cable into the white connector on the back of the drive unit. Then connect the numbered end of the wide gray signal cable to its connector on the disk drive *(right)*.



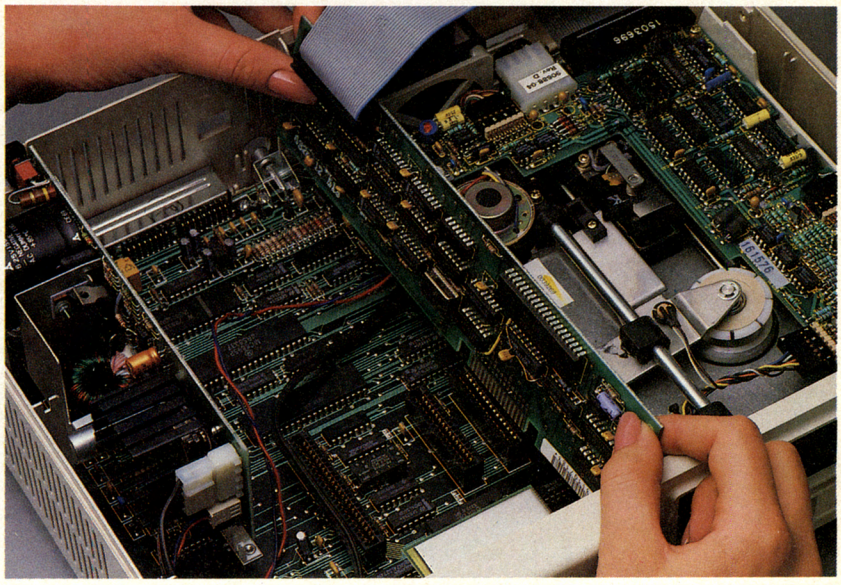
To install the disk drive, first snap out the plastic disk face plate from the inside of the front panel of your PCjr. With the front of the computer facing you, angle the disk drive unit into place as shown at right, holding the black disk-drive power cable and the red-and-blue fan power cable so that they run under the drive unit and emerge on the other side. Position the disk drive in the computer's front panel, and align the white mounting pins on the bottom of the drive with the white holes in the system unit. Press the disk drive down until it snaps into place.



Bring the power cables for the disk drive and the fan across the bottom of the system unit, toward the power board on the left side; take care to keep both cables flat. Plug the red-and-blue fan power cable into the lower of the two white connectors on the right edge of the power board. Then plug the black disk-drive power cable into the upper white connector *(right)*.

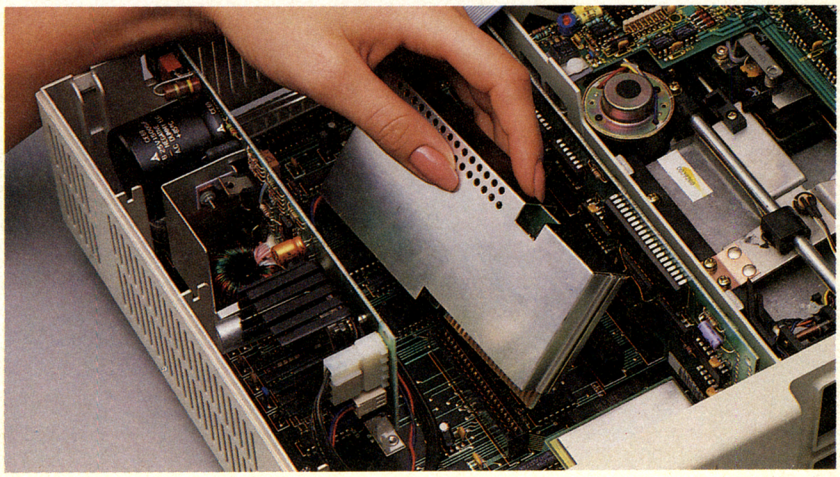


Find the disk-drive adapter board that is packaged along with the drive unit. Plug the free end of the gray signal cable into the connector on the top left side of the board. With both hands, lower the adapter board's connector into the first slot beside the drive unit, as shown at right. Make certain that the left side of the adapter board slides into the first of the two plastic guides located along the side of the system unit.



A further enhancement for the PCjr is a 64KB Memory and Display Expansion Option, which adds the extra memory required to run many of the programs available for the PCjr. Guide the connector of the memory expansion board, encased in its protective metal holder, into the slot beside the power board *(right)*.

Now you may replace the cover of the system unit. Before running any programs, check the disk drive by running the internal diagnostic tests contained in the PCjr's memory *(pages 24-25)*.



Bibliography

Books

Cobb, Douglas Ford, and Chris DeVoney, *Introducing the IBM PCjr.* Que, 1983.

Disk Operating System (IBM), Microsoft, 1983.

Goldstein, Larry Joel, *IBM PCjr Buyer and User Guide.* Brady, 1984.

IBM Personal Computer PCjr Hardware Reference Library:
Guide to Operations. IBM, 1983.
Technical Reference. IBM, 1983.

Jordan, Larry E., and Bruce Churchill, *Communications and Networking for the IBM PC.* Brady, 1983.

Kruglinski, David, *Data Base Management Systems: A Guide to Microcomputer Software.* Osborne/McGraw-Hill, 1983.

Laurie, Peter, *The Joy of Computers.* London: Hutchinson, 1983.

Seybold, Andrew M., *Introducing the IBM PCjr: Unshelling the Peanut.* Sams, 1983.

Magazine Articles

"The Computer as a Teacher." *Consumer Reports,* November 1983.

"Demystifying Computer Languages." *Popular Computing,* September 1983.

D'Ignazio, Fred, "Smarter with Age." *PC World,* September 1983.

Frankel, David, and Michael Guttman, "Getting to First Base." *PC World,* May 1983.

Freedman, Eric, "The Many Flavors of The Source." *PCjr,* April 1984.

Freff, "Making Music with the Well-Synthesized PC." *PC Magazine,* December 1983.

Gabel, David, "What's in a Game?" *Personal Computing,* April 1983.

Helmers, Carl, "Turtles." *Robotics Age,* May/June 1983.

Immel, Richard A., "Bad Business Software." *Popular Computing,* November 1983.

Jenkins, David, "Going for the Record." *PC World,* November 1983.

Levering, Robert, "Data Management without Programming." *PC World,* April 1983.

Martin, Janette, "Dial Up News and Entertainment." *PC World,* April 1984.

Rietmann, Kearney, "PCjr World." *PC World,* February 1984.

Sandler, Corey, "Homeword Bound." *PCjr,* February 1984.

Seger, Katie, and Frederic E. Davis, "IBM's Home Run." *PC World,* February 1984.

Seger, Katie, and Adrian Mello, "Charting the PCjr Software Course." *PC World,* February 1984.

Shea, Tom, "Personal Computer Graphics: Pushing Technology to the Limit." *InfoWorld,* November 1983.

Steffin, Sherwin, "What Does the Computer Teach Best?" *Softline,* March 1982.

Williams, Linda V., "No More Pencils." *PC World,* November 1983.

Magazines

Compute!'s PC & PCjr, March 1984.

The Guide to Personal Computer Offerings from IBM. Fall 1983/Winter 1984.

Money Guide to Personal Computers, 1984.

PCjr, February-March 1984.

PC World, November 1983-February 1984.

PC World Annual Software Review, 1983/1984.

Picture Credits

Sources for all pictures in this book are shown below. Credits for the illustrations from top to bottom are separated by dashes.

The following photographs are by Larry Sherer: pages 3-8, 11-17, 19 bottom-22 top, 23 bottom-30, 36 bottom, 44-50, 51 top and center, 52-55, 56 bottom-61, 65-69, 74, 93-99.

The following photographs are courtesy Spinnaker Software Corp.: 51 bottom and 56 top.

All drawings are by Matt McMullen with the exception of pages 22 center and 76 top: William J. Hennessy.

Computer screen images were generated courtesy the following sources: page 4: Internal Diagnostics, International Business Machines (IBM). 20: IBM. 22 top: Machine BASIC, IBM. 23 bottom: Keyboard Adventure, IBM. 24, 25: Internal Diagnostics, IBM. 33-43: Disk-Operating System (DOS) 2.1, IBM. 44: Chess 1.3, PCsoft Consulting Inc. 46: My Letters, Numbers and Words by Elmer Larsen, Stone & Associates, Inc. — Bumble Game, The Learning Company and IBM. 47: Adventures in Math, IBM — Monster Math, IBM. 48: Bumble Plot, The Learning Company and IBM — Speed Reader II, Davidson & Associates. 49: Computer SAT, Harcourt Brace Jovanovich. 50: Mine Shaft, Sierra-on-Line and IBM — Quad, Acorn Software. 51: Casino Games, IBM — Adventures in Serenia, Sierra-on-Line and IBM — Trains™, Spinnaker Software Corp. 56: Facemaker™, Spinnaker Software Corp. — Delta Drawing™ Learning Program, Spinnaker Software Corp. 57: PC Design, Koala Technologies Corporation. 61: Homeword, Sierra-on-Line and IBM. 63: PFS:File, Software Publishing Corporation and IBM. 65: VisiCalc 1.1, IBM. 71: PCjr Sampler, IBM. 72, 73: CompuServe Information Service. 76: Cartridge BASIC, IBM.

All other illustrations and screen images were created by Time-Life Books, with the assistance of the authors and consultants.

Acknowledgments

The editors are particularly indebted to Robert P. Bartosak, consultant, for his help in the preparation of this book.

Index with Glossary

Numerals in italics indicate an illustration of the subject mentioned.

A Adapter: card, for modem, *68;* color/graphics monitor, 93
Address: *a storage location in memory. It can hold numeric or alphabetic data or a program instruction.*
Application programs, 31; business graphics, 59; data base management, 59, 62-63; loading, 32; self-loading disk, 31, 43, 47, 59; spreadsheet, 59, 64-65; word processing, 59, 60-61
Artificial intelligence, 53
ASCII code, 25, 91

B Backup: *a duplicate copy of stored information on a disk;* 39, 47, 97
BASIC, 21, 22, 31, 53; Cartridge, 21, 26-29, 76-77; Cassette, 26-27, 76; commands, 78-79; compiler, 75; equal sign in, 82, 84; error messages, 96; function keys, *76;* graphics programming, 92-93; loops, 86-87, 88, 91; mathematical computations, 79, 82-83; programming in, 75-95; programming mode, 76, 77; sound programming, 92-93; source code, 75, 86; statements, 79; subroutines, 90-91, 94, 95; variables, 80-83
Baud rate, 70
Block: *a sector of disk storage that holds 512 characters.*
Booting: *the process of putting a program in control of the computer, also called system reset;* 32, 33
Bug: *a software error in a program;* 94
Bulletin board (electronic), 67
Bus: *a set of electrical wires inside the system unit, connecting the computer's components.*
Byte, 11, 22. *See also Useful Terms*

C Cables, 11, 12, *18, 19;* adapter, for RGB monitor, 14; for disk drive, *98,* 99; linking keyboard to system unit with, *13;* printer, *17*
Cartridges, *3,* 11; inserting, *28. See also Useful Terms*
Cassette recorder, 11, 26
Cassette tape, 10, 26
Chip: *a miniaturized electronic circuit on a silicon flake 1/16" to 3/8" square.*
Cls command, 78, 79
Color: capabilities, 21; in Cartridge BASIC, 29; in Cassette BASIC, 27; programming, 93
Color statement, 93
Communications: between computers, 67, 70, 71; installing modem, *68-69;* with mainframe, 67, 70; program, 70, 71; subscribing to information service, 67, 72-73
Components: adapters for optional equipment, 9; care of, 97; connecting, *11, 12-19;* diagnostic tests, 24-25; entry-level system, 11, *18;* expanded system, 9, 11, *19;* placement, 97; setting up, 9
CompuServe, 72, 73
Controller, for games, 50
CPU (central processing unit), 10, 12
Cursor: *a flashing marker on a monitor screen, showing where the next character will be displayed;* 21

D Data. *See Useful Terms*
Data base: *a large file of information on a single subject or related subjects. A data base is continually updated and is organized for easy access;* 62, 67
Data base management, programs, 59, *62-63*
Device name, 38, 39
Diagnostic routines, 21, 24-25
Directory: *a list, or table of contents, of programs or data files stored on a disk.* Displaying, 34-35
Disk drive, 10, *11, 19,* 26; adding to entry-level machine, *98-99;* DOS and, 32, 33
DOS, 31, 32-33; backup copy, 39; booting, 32, 33; commands for handling files, 42-43; copying files, 38-39, 40; copying files from keyboard, 41; copying programs, 32; formatting disk, 36-37; locating files, 34-35; making a program self-loading, 43, 47; naming files, 35; storing files, 34

E Educational software, 45, 46-49; choosing, 45, 46, 48, 49; graphics software, 56; monitor for, 48; programming in Logo, 52-55
Electrical precautions, 19, 97
Electronic mail, 67, 73
End statement, 79, 90
Entertainment. *See* Games
Entry-level model, 2, 11, *18;* adding accessories, *98-99*
Error messages, 96
Expanded system, 2, 9, 11, *19*
Expansion board, 10, 59; adding to entry-level machine, *99*
Expansion slot, *68, 69*

F Files: changing name, 43; copying on disks, 38-39, 40; copying from keyboard, 41; displaying, 43; DOS commands, 42-43; formatting new disk, 36-37; locating, 34-35, 37; naming, 34, 35; storing on disks, 34, 36; storing information on disks (data base management), 62-63. *See also Useful Terms*
Filespec, 35, 38, 39
Floppy disks, *3,* 10; care of, 36, 97; copying files, 38-39, 40; formatting new, 36-37; loading, 78; naming files, 34, 35; program, 31; storing files on, 34; storing program on, 78. *See also Useful Terms*
Flow chart, *86, 90*
For . . . next statement, 79, 86-87, 91, 93; *chart* 79
Format: *a magnetic pattern recorded on a disk to organize it by track and sector, for easy storage and retrieval of files; 36,* 37, 47
Formatting, 36-37

G Games: joystick, *3,* 50; role of, 45; software, 45, 50-51
Gosub statement, 79, 90, 91
Goto statement, 79, 84, 85
Graphics: business, 59; capabilities, 21, 27; in Cartridge BASIC, 28, 29, 76; in Cassette BASIC, 27; characters, 91; drawing with touch pad, *57;* programming, 92-93; software, 56; statements, 93

H Hard copy: *paper copy of any computer output, including programs or data, made by a printer or plotter connected to a computer;* 27
Hardware, 31. *See also* Components; *Useful Terms*
Home office, 59; setup, *19*

I *If . . . then* statement, 79, 84-85, 89, 93
Immediate (command) mode, 77, 78
Information services: subscribing to, 67, 72-73; utilities, 73
Infrared transmission, 12, *13,* 18
Input, 12, 13, 32. *See also Useful Terms*
Input device, 31, 39
Input/output slots, *11*
Input statement, 79, 80, 81, 82, 87, 93
Integrated software, 59
Interface: *a device that translates signals between two computers or accessories to send or receive data accurately; 17*
Interpreter: *a computer language that, each time a program is executed, translates each statement in it into a form the machine can perform directly;* 75, 76, 77, 78, 79

J Joystick, *3, 18,* 27; discrete, 50; for games, 50; proportional, 50

K Keyboard, *5, 12;* BASIC function keys,

76; cable for linking to system unit, 12, *13;* infrared transmission, 12, *13,* 18; setting up, *18, 19;* special keys, 21, *22, chart* 23; tutorial, 21, 23

L Language: *an organized set of instructions used to communicate with a computer. Low-level languages, such as assembly language, use the computer's built-in machine language. High-level languages, such as BASIC, Logo, COBOL and Pascal, use English-like commands, which a compiler or interpreter must translate before or during use;* 21, 53. BASIC, 26-29, 75, 76, 77; built-in, 22; compiled, 75; interpretive, 75; Logo, 52, 53
Let statement, 79, 82-83
Light pen: *a light-sensitive pointer whose position on the monitor screen can be read by the computer with the appropriate software;* 27
List command, 76, 78, 79
Load command, 76, 78, 79
Locate statement, 79, 91
Logo, 49, 52-55
Loops, 86-87, 88, 91, 93

M Memory, 10, 11, 22; adding more with expansion board, 59, *99. See also Useful Terms*
Menu: *a screen display listing program options, each identified by a number or letter. The user selects the desired option by pressing the appropriate number or letter key.*
Modem: *a telephone interface that converts a computer's digital signals to signals that can be sent over phone lines. There are acoustic and direct-connect modems;* 26. Acoustic, *69;* direct-connect, 69; external, 68, *69;* internal, *68-69*
Monitor: cable for, 14; color/graphics adapter, 93; composite video, *4,* 14, 48; connecting, *14;* for educational programs, 48; 80-column display, 62; and graphics, 14; RGB, *14;* setting up, 18, *19;* and word processing, 14
Music: capabilities, 21; in Cartridge BASIC, 28, 29, 76; in Cassette BASIC, 26; programming, 92

N Network: *a method of connecting several computers to share common accessories and/or data. See* Communications
New command, 78, 79

O Operating system, 31. *See also* DOS; *Useful Terms*
Outlet bar (box), 18, *19,* 22
Output. *See Useful Terms*
Output device, 31, 39

P Parallel printer adapter, *17*
Peripheral: *an accessory connected to a computer;* 2
Pixel: *one graphic point (a picture element) on the video display.*
Play statement, 92
Plotter: *a computer-controllable drawing device that creates pictures or graphs on paper.*
Port: *a connector in a computer, through which input and output data passes to or from a peripheral; 16,* 17, 68, *69*
Power supply. *See* Transformer
Printer: *an accessory that produces a paper copy of computer output. Daisy-wheel printers produce full-formed characters; dot-matrix printers make characters from patterns of dots; 3,* 27. Adapter for other models, *17;* connecting, *16,* 17; daisy-wheel, 17; dot-matrix, 17; inserting paper, *16;* letter-quality, 17; setting up, *18, 19*
Print statement, 26, 79, 80, 81, 88, 89, 93
Program, 22, 31; communications, 70, 71; incompatible, 42; loading into memory, 78; storing on disk, 78; writing sample, 94-95. *See also Useful Terms*
Programming: in BASIC, 75-95; branching a program, 84-85; correcting errors, 78-79; flow-chart, *86, 90;* graphics, 92-93; in Logo, 52-55; loops, 86-87; manipulating variables, 80-81; mode, 76, 77, 78; simple commands, 78-79; sound, 92; subroutines, 90-91, 94, 95

R RAM (random access memory), 10. *See also* Read-and-write memory; *Useful Terms*
Read: *a computer operation to bring information into memory from a storage device or a keyboard;* 10, 32
Read-and-write memory, 10, 11, 27, 76
Read/data statement, 79, 88-89
Restore statement, 89
Return statement, 79, 90, 91
RF modulator, 14, *18*
ROM (read-only memory), 10, 11, 21, 26, 27, 76. *See also Useful Terms*
Run command, 76, 77, 78, 79

S *Save* command, 76, 78, 79
Screen statement, 92-93
Software, 31; application programs, 59-65; educational, 45, 46-49; games, 50-51; graphics, 56; integrated, 59; Logo, 52-55. *See also Useful Terms*
Sound: capabilities, 21; in Cartridge BASIC, 28, 29; in Cassette BASIC, 26; programming, 92; statement, 92
Source, The, 72, 73
Source code, 75, 86
Spreadsheet: *a program for financial planning, income tax preparation, budgets or other functions based on equations. It appears to the user as an accountant's spreadsheet.* Programs, 59, 64-65
Static electricity, 9, 97
String: *any group of alphanumeric characters stored in computer memory and handled as a unit;* 92. Variables, 80, 81, 82, 83, 88
Subroutines, 90-91, 94, 95
Syntax: *the correct grammatical form for computer instructions;* 78
Syntax error, 22, 76, 78-79
System unit, *5, 10;* enhanced, 9, *10,* 11, *19;* entry-level, 2, 11, *18;* linking to keyboard, 12, *13;* setting up, *18, 19*

T Television: connecting to PCjr, *15;* as monitor, *4,* 14; RF modulator, 14; setting up, *18*
Thermal paper, *3, 16,* 17
Toggle. *See Useful Terms*
Touch pad, *57*
Transformer, *5,* 10, 11, *18, 19;* connecting, *11*
Turtle graphics, 52-53. *See also* Logo
Tutorial: educational software, 46, 48, 49; Keyboard Adventures, 21, 23; SAT, 49

U *Useful Terms,* inside front cover
Utility program: *a computer program that helps the programmer carry out frequently performed tasks, such as scanning a disk directory, making copies of disks or formatting new disks.*

V Variables, 80-83, 84, 86, 87, 88, 89, 92, 93; numeric, 80, 81, 82, 88; string, 80, 81, 82, 83, 88

W Word processor: *a program used in composing, editing, storing or duplicating letters or other text pieces. Also, the hardware used to carry out these activities.* Monitor, 14; printer, *17;* program, *36-37;* software, 59, 60-61
Write: *a computer operation to copy information to a storage or display device from computer memory;* 10, 32
Write-protect notch: *a safety notch in a disk or cassette that, when covered, prevents erasure of important information; 36*